



DYNAMIC PROTOCOL REVERSE ENGINEERING
A GRAMMATICAL INFERENCE APPROACH

THESIS

Mark E. DeYoung, Captain, USAF

AFIT/GCS/ENG/08-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

DYNAMIC PROTOCOL REVERSE ENGINEERING
A GRAMMATICAL INFERENCE APPROACH

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science (Computer Science)

Mark E. DeYoung, B.S.

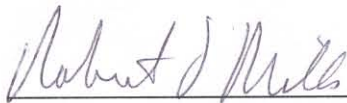
Captain, USAF

March 2008

DYNAMIC PROTOCOL REVERSE ENGINEERING
A GRAMMATICAL INFERENCE APPROACH

Mark E. DeYoung, B.S.
Captain, USAF

Approved:



Dr. R. Mills, (Chairman)

26 Feb 08

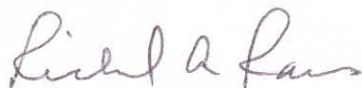
Date



Lt Col S. Kurkowski, USAF, PhD
(Member)

26 Feb 08

Date



Dr. R. Raines (Member)

26 Feb 08

Date

Abstract

Round trip engineering of software from source code and reverse engineering of software from binary files have both been extensively studied and the state-of-practice have documented tools and techniques. Forward engineering of protocols has also been extensively studied and there are firmly established techniques for generating correct protocols. While observation of protocol behavior for performance testing has been studied and techniques established, reverse engineering of protocol control flow from observations of protocol behavior has not received the same level of attention. State-of-practice in reverse engineering the control flow of computer network protocols is comprised of mostly ad hoc approaches. We examine state-of-practice tools and techniques used in three open source projects: Pidgin, Samba, and rdesktop. We examine techniques proposed by computational learning researchers for grammatical inference. We propose to extend the state-of-art by inferring protocol control flow using grammatical inference inspired techniques to reverse engineer automata representations from captured data flows. We present evidence that grammatical inference is applicable to the problem domain under consideration.

Acknowledgements

My sincerest thanks and love go to my wife and son for enduring the rigors of academic research over the last eighteen months. I would like to express my sincere appreciation to my faculty advisor, Dr. Robert Mills, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. He allowed me to approach the topic at a grandiose scale and gently reined me in to an achievable and interesting scope.

Mark E. DeYoung

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	x
List of Tables	xii
List of Symbols	xiv
List of Abbreviations	xv
I. Introduction	1
1.1 Operations in Cyberspace	1
1.2 Problem Domain	2
1.3 Related Problem Domains	3
1.4 Related Application Domains	4
1.5 Investigative Questions	4
1.6 Document Overview	5
II. Problem Domain	6
2.1 Distributed Systems	6
2.1.1 Data Structures	9
2.1.2 Data Relationships	10
2.1.3 Other Data Characteristics	13
2.2 Network Trace Collection	14
2.3 Application Level Protocol Data Flow Recovery	16
2.3.1 Naïve Flow Membership	16
2.3.2 Accurate Flow Membership	16
2.3.3 Stateful vs. Stateless	17
2.3.4 Single-connection vs. Multi-connection	18
2.3.5 Single-channel vs. Multi-channel	18
2.4 Application Level Network Traces into Automata	19
2.5 Protocol Design Recovery	20
2.5.1 Forward Engineering	20
2.5.2 Reverse Engineering	21
2.5.3 Protocol Reverse Engineering	21
2.6 State-of-practice	23

	Page
2.6.1 Tools	24
2.6.2 Techniques	27
2.7 Case Studies	28
2.7.1 Pidgin	28
2.7.2 Samba	29
2.7.3 Rdesktop	31
2.8 State-of-Art	32
2.8.1 Protocol Format Recovery	33
2.8.2 Recovering Automata	34
2.9 Chapter Summary	37
III. Algorithm Domain	39
3.1 Design Recovery from Samples of Behavior	39
3.2 A Language Recognition Problem?	39
3.3 Formal Languages	40
3.3.1 Chomsky Hierarchy	40
3.3.2 Other Formal Reperesentations	44
3.4 Learning Automata Representation of a Language	45
3.5 Computational Learnability Models	45
3.5.1 Identification in the Limit	46
3.5.2 Query Learning Model	47
3.5.3 PAC Learning Model	47
3.6 Tractability	48
3.7 Search Approaches for Grammar Induction	48
3.8 Notations and Definitions	49
3.9 Grammatical Inference Algorithms	54
3.10 Inference of Regular Languages (Type-3)	55
3.10.1 Statistical Extrinsic Methods	56
3.10.2 Extrinsic Negative Sample Support Methods	57
3.10.3 Characterizeable Methods	59
3.10.4 Heuristic Methods	61
3.10.5 Hybrid Methods	63
3.11 Inference of Higher Order Languages	64
3.12 Chapter Summary	65
IV. Experimental Design	66
4.1 Application Level Network Traces into Automata	66
4.2 Protocol Selection	66
4.3 Algorithm Selection	67
4.4 Experimental Architecture	67

	Page
4.4.1 Network Trace Collection	69
4.4.2 Application Level Protocol Data Flow Recovery	69
4.4.3 Protocol Format Recovery	69
4.4.4 Protocol Transition Function Recovery	69
4.5 Limiting Factors	70
4.5.1 Session Detection	70
4.5.2 Online vs. Offline Analysis	71
4.5.3 Target Automata Representation	71
4.5.4 Incomplete Data	71
4.5.5 Noisy Data	73
4.5.6 Connection Level Protocol Stack	77
4.6 Extrinsic Heuristics for Noise Filtering	77
4.7 Inference Accuracy	79
4.7.1 Inferred POP3 Control Flow	80
4.7.2 Inferred SMTP Control Flow	82
4.8 Sensitivity to Noise	84
4.9 Algorithm Runtimes	87
4.10 Chapter Summary	88
V. Analysis and Results	89
5.1 Conclusions	89
5.1.1 Experimental Results	89
5.1.2 Investigative Questions Answered	89
5.2 Future Work	90
5.3 Summary	92
Appendix A. Data	93
A.1 Natural Data Sets	93
A.2 Artificial Data Sets	93
A.3 DARPA Intrusion Detection Evaluation data set	93
A.3.1 IDEVAL Data Quality	94
A.3.2 IDEVAL Data Relevance	94
A.4 Data Files	94
A.4.1 Complete Data File set	95
A.4.2 Merged Data File set	98
A.5 SMTP Sample Data	101
A.6 POP3 Sample Data	109

	Page
Appendix B. Low Level Implementation	114
B.1 Sources for Protocol Formats	114
B.2 Automata Toolkits	115
B.2.1 AMoRE	115
B.2.2 Vaucanson	115
B.2.3 JFLAP	115
B.2.4 Grail	115
B.3 Grammatical Inference Implementations	116
B.3.1 LearnLib	116
B.3.2 Mical	116
B.3.3 Other Implementations	117
B.4 Implementation Language	117
B.5 Low Level Implementation	117
B.5.1 flowtool	117
B.5.2 flowinfer	118
Appendix C. Inferred Automaton	121
Bibliography	128

List of Figures

Figure		Page
2.1	OSI Reference Model.	7
2.2	Internet from TCP Perspective.	8
2.3	TCP Client/Server Topology.	9
2.4	IP Packet Structure.	9
2.5	UDP Packet Structure.	10
2.6	TCP Packet Structure.	11
2.7	Trace Collection Architecture.	14
2.8	Bro deployment with network tap.	15
2.9	Wireshark Following SMTP Conversation.	17
2.10	Flow level breakdown of a simple FTP transfer.	19
2.11	Relationship of Re-engineering Practices.	21
2.12	Wireshark - Ready for “test, capture, and stare”.	26
2.13	MITM Attack Topology.	28
2.14	SAMBA on Macintosh OS X 10.	30
2.15	Rdesktop Connection.	32
3.1	Chomsky Hierarchy.	42
3.2	State Diagram for SMTP Sender.	44
3.3	Gold’s Enumeration Procedure.	46
3.4	Example PTA.	53
3.5	Some Families of Regular Languages.	60
4.1	Experimental Architecture Overview.	68
4.2	Wireshark Following Bad SMTP Session.	76
4.3	SMTP Session Initiation.	80
4.4	SMTP Session Transaction.	81
4.5	SMTP Session Termination.	81

Figure		Page
4.6	POP3 Session Initiation.	81
4.7	POP3 Session Transaction.	81
4.8	POP3 Session Termination.	82
A.1	IDEVAL 1999 Network Topology.	95
A.2	Experimental Architecture Pre-Processing.	99
B.1	Flowtool verbose sample output.	118
B.2	Flowtool call graph (elided).	119
C.1	k -TSSI POP3 Composite Final Automaton $k = 1$	122
C.2	k -RI POP3 Composite Final Automaton $k = 1$	123
C.3	k -RI POP3 Composite Final Automaton $k = 2$	124
C.4	k -TSSI POP3 Composite Final Automaton $k = 2$	125
C.5	k -RI POP3 Composite Final Automaton $k = 3$	126
C.6	k -TSSI POP3 Composite Final Automaton $k = 3$	127

List of Tables

Table		Page
3.1	Chomsky Hierarchy.	42
3.2	SMTP Sender Transitions.	43
3.3	Regular Inference Algorithms.	55
3.4	Regular Inference Algorithm Performance.	56
3.5	Muggleton Predicate Functions $\chi(u, v)$ for k -tails.	57
4.1	POP3 Command Alphabet Weekly Overview.	72
4.2	SMTP Cumulative Command Alphabet.	73
4.3	SMTP Command Alphabet Weekly Overview.	74
4.4	SMTP Reply Alphabet Summary.	75
4.5	SMTP Reply Alphabet Weekly Overview.	75
4.6	Buffer overflow attacks on SMTP server at 172.16.114.50. . . .	78
4.7	k -RI POP3 Composite Automaton Filtered.	82
4.8	k -TSSI POP3 Composite Automaton Filtered.	83
4.9	k -RI POP3 Composite Automaton Unfiltered.	83
4.10	k -TSSI POP3 Composite Automaton Unfiltered.	84
4.11	k -RI SMTP Composite Automaton Filtered.	85
4.12	k -TSSI SMTP Composite Automaton Filtered.	85
4.13	k -RI SMTP Composite Automaton Unfiltered.	86
4.14	k -TSSI SMTP Composite Automaton Unfiltered.	86
4.15	k -RI SMTP Composite Automaton Type-41 removed.	87
4.16	POP3 Composite Runtimes.	88
4.17	SMTP Composite Runtimes.	88
A.1	IDEVAL Data Files Size.	96
A.2	IDEVAL Data Files Time.	97
A.3	Merged Data Files Size.	100

Table		Page
A.4	Merged Data Files Time.	100
A.5	SMTP TCP Connection Summary.	101
A.7	Data Summary: SMTP Command Alphabet total.pcap.	101
A.8	Data Summary: SMTP Reply Alphabet total.pcap.	103
A.9	Data Summary: SMTP Composite Alphabet total.pcap.	104
A.10	POP3 TCP Connection Summary.	109
A.12	Data Summary: POP3 Command Alphabet total.pcap.	109
A.13	Data Summary: POP3 Reply Alphabet total.pcap.	110
A.14	Data Summary: POP3 Composite Alphabet total.pcap.	111

List of Symbols

Symbol		Page
\mathcal{L}	A class of languages	49
Σ	A finite alphabet	49
δ	A partial mapping from $S \times \Sigma \rightarrow S$	50
$Pref_A(q)$	The prefix language of a state q in automaton A	50
$Suff_A(q)$	The suffix language of a state q in automaton A	50
$Acc_A(w)$	The set of acceptances of a word w in automaton A	50
π_S	Partition of set S	51
$B(s, \pi_S)$	Block of π_S containing s	52
$s\mathcal{LT}$	Class of strictly locally testable languages	54
$Pre(L)$	The set of all prefixes of elements of language L	54
$w \setminus L$	left-quotient of language L and word w	54
$w \setminus^k L$	The k -tails of word w in language L	54
\mathcal{Rev}_k	Class of k -reversible languages	54

List of Abbreviations

Abbreviation		Page
USAF	United States Air Force	1
GI	Grammatical Inference	4
OSI	Open Systems Interconnect	6
ISO	International Organization for Standards	6
TCP	Transmission Control Protocol	7
IP	Internet Protocol	7
IPv4	TCP/IP protocol family version 4	7
FSM	Finite State Machine	8
UDP	User Datagram Protocol	9
DNS	Domain Name Service	10
ICMP	Internet Control Message Protocol	11
ARP	Address Resolution Protocol	11
RIP	Routing Information Protocol	11
OSPF	Open Shortest Path First	11
IPsec	IP security	12
VPN	Virtual Private Network	12
IS-9646	International Standard 9646	14
USAP	Upper Service Access Points	14
LSAP	Lower Service Access Points	14
RCE	reverse code engineering	21
SMB/CIFS	Server Message Block/Common Internet File System . . .	23
API	application programming interface	24
MITM	Man in the Middle	27
RPC	Remote Procedure Call	34
MSC	Message Sequence Charts	35

Abbreviation		Page
CFSM	Communicating Finite State Machine	36
BNF	Baukus-Naur Form	41
SMTP	Simple Mail Transfer Protocol	43
PAC	probably approximately correct	46
FSA	Finite State Automaton	49
DFA	Deterministic Finite State Automaton	50
CA	Canonical Automaton	51
UA	Universal Automaton	51
MCA	Maximal Canonical Automaton	51
PTA	Prefix Tree Acceptor	52
APTA	Augmented Prefix Tree Acceptor	52
RIG	Regular Inference of Grammars	58
BRIG	Boosted beam-search Regular Inference of Grammars . . .	58
RPNI	Regular Positive and Negative Inference	58
EDSM	Evidence Driven State Merging	59
k -TSSI	k -Testable in the Strict Sense of Inference	59
k -RI	k -Reversible Inference	61
MGGI	Morphic Generator Grammatical Inference	61
ECGI	Error Correcting Grammatical Inference	63
NTS	Non-terminally Seperated Languages	65

DYNAMIC PROTOCOL REVERSE ENGINEERING

A GRAMMATICAL INFERENCE APPROACH

I. Introduction

As the United States Air Force (USAF) extends into the Cyberspace domain, the ease of breaking into computer networks and misusing distributed systems has become increasingly problematic [163, 172, 173, 242, 280]. Deep understanding of the protocols which traverse computer networks and enable distributed systems is increasingly important to securing our computer networks and putting opponent's networked operations at risk.

1.1 Operations in Cyberspace

The DOD defines Cyberspace as a domain characterized by the use of electronics and the electromagnetic spectrum to store, modify, and exchange data via networked systems and associated infrastructures. Operations in Cyberspace have both strategic and tactical requirements. Tactics, Techniques and Procedures coupled with weapons systems that produce reliable and predictable battle effects are essential. Freedom of Cyberspace much like Freedom of the Seas and Freedom of the Skies has become essential to our way of life. As such, our current inability to operate in Cyberspace as a domain of military operations, governed by mathematical and electromagnetic principles, requires us to develop, train and equip cyber forces that can guarantee Freedom of Cyberspace. In the words of Secretary Wynne [280]:

Cyberspace is a domain for projecting and protecting national power, for both strategic and tactical operations [212].

Vulnerabilities in technical standards and concrete implementations of technical standards are cyber warriors fighting positions. Freedom of Cyberspace will require tactical cyber power: the ability to degrade, disrupt, deny and destroy adversaries

fighting positions while defending our own. Deep understanding of distributed system is a critical enabler to developing cyber power in network centric cyber spaces.

In this thesis we focus on protocol reverse engineering as a method that enables the generation of instruments of tactical cyber power in digital computer networks. We do not discuss the broader topics of computer network exploitation/protection or electronic warfare. In fact, we view protocol reverse engineering as only one facet of the larger topic of tactical cyber power. Also, we will concentrate on technical means that enable creation of tactical cyber weapons over doctrine, organization, and policy.

At the outset of performing the preliminary literature review it was apparent that the volume of academic information concerning protocol forward engineering greatly exceeded the volume of academic information on protocol reverse engineering. It is our contention that the state-of-the-art in protocol reverse engineering methods and tools remains largely shrouded from the view of the general public.

Due, in part, to the underground nature of the subject, effective application of protocol reverse engineering to generate effective instruments of tactical cyber power is a challenging problem.

Generation of tactical cyber weapons requires a deep understanding of the technical architecture of the systems under consideration. A cyber weapon must provide effective, reliable, and repeatable, battlespace effects.

A first step is to recover models of protocols that increase analyst understanding and support formal analysis to verify the effects of network centric cyber weapons.

1.2 Problem Domain

Correct protocol design is a difficult engineering task. Gerard Holzmann offers the following:

It is the unexpected sequences of events that lead to protocol failures, and the hardest problem in protocol design is precisely that we must try to expect the unexpected [109].

Protocol model recovery from network traffic is a challenging problem.

! While the algorithm domain under consideration is proveably **NPC** we do not provide proof that the problem domain is **NPC**. We only conjecture the problem domain is **NPC**.

Given the complexity of correctly designing a protocol specification and then accurately engineering a protocol implementation it is not surprising that protocols exhibit vulnerabilities. Causes of unexpected conditions that expose vulnerabilities range from accidental oversight [220] to deliberate attack [12, Section 7.2].

The problem domain under consideration is design recovery of protocol models from captured data flows. Ultimately, the recovered designs should support formal analysis that identifies implementation issues that allow deliberate attack or accidental failure. In essence, can we discover protocol implementation issues that allow deliberately crafted packets which lead a protocol parser to unexpected conditions?

1.3 Related Problem Domains

Network traffic classification and deep packet inspection are related to protocol reverse engineering. Both domains require understanding of protocols that might not be documented in open specifications [76, 126, 127, 185, 193, 200, 264]. Likewise, signature based intrusion detection requires deep knowledge of protocols' inner workings [178]. We conjecture that behavioral based intrusion detection could also benefit from models recovered via protocol reverse engineering. Finally, protocol reverse engineering can draw practical methods from the domain of protocol conformance testing.

While we concentrated on a subset of application level protocols on IPv4 networks similar experimental analysis could be conducted against other classes of protocols, such as SCADA ¹ or SS7 ², for vulnerability assessment and potentially generation of targeted effects.

¹Supervisory Control and Data Acquisition - [47, 79, 134] introduce the subject. [47, Section 6.7, Chap 12] covers TCP/IP encapsulated SCADA communications.

²Signaling System 7 - a set of telephony signaling protocols

1.4 *Related Application Domains*

Automated specification generation, automated test generation and automated conformance testing provide architectures that are useful for protocol reverse engineering. Dssouli et al presents a test automation architecture for distributed systems in [68]. Ammons examines automated specification generation from program execution traces [6]. Automated test generation for white-box and black-box testing is well studied for software testing. Random boundary testing methods for network protocols are covered by [249, Chapter 14] and for software testing in [175, 188].

Tretmans covers OSI protocol conformance testing in [257] while Berg examines the similarities between regular inference and conformance testing in “On the Correspondence Between Conformance Testing and Regular Inference” [22]. Conformance testing as an Angluin query styled learning problem is also examined by Lai in [136] which presents a genetic algorithm approach to adaptive model checking.

1.5 *Investigative Questions*

The focus of this research is the evaluation of existing Grammatical Inference (GI) algorithms for the dynamic protocol reverse engineering domain. GI is a branch of artificial intelligence that concentrates on the inference of grammatical structures from samples of a language. In particular, we ask the following questions:

IQ1 What information is necessary to reverse engineer the control portion of application layer protocols from data flows?

IQ2 Given the proven [7, 95] difficulty of inferring finite automata from positive samples only, are there GI approaches that are appropriate for reverse engineering automata representations of the control portion of application layer protocols from data flows?

We propose to apply GI algorithms to recover structure from the protocol stream that is not immediately obvious from observation of individual packets. We hope to advance the state-of-art in protocol reverse engineering by automatically revealing

structural relationships much like an oscilloscope displays waveforms from an electric signal. We will concentrate on *a posteriori* analysis instead of *online* analysis of live execution traces.

While we discuss computer science oriented theoretical aspects of GI we are more interested in the engineering oriented pragmatic recovery of structure and the presentation of experimental evidence that establishes the applicability of GI to the problem of protocol model recovery.

Finally, we recognize that the approach presented is only a partial solution to the problem domain under consideration. Human analysts must continue to apply common heuristics (e.g. identifying signpost values, block structure inference, or windowed entropy).

1.6 Document Overview

In Chapter II we present the problem domain under consideration. Next, in Chapter III we introduce the Chomsky Hierarchy as a framework for discussing computational learnability and overview several existing grammatical inference algorithms. In Chapter IV we describe the experimental architecture used to evaluate two selected grammatical inference algorithms against POP3 and SMTP traffic from the IDEVAL data set. Finally, in Chapter V we provide the results of our evaluation and propose areas that can be refined in future work.

II. Problem Domain

This chapter provides background regarding the problem domain. First, we describe the problem domain under consideration with an English description. We discuss distributed systems and issues related to protocol design recovery. Finally, we introduce protocol reverse engineering and overview state-of-practice and state-of-art reverse engineering tools and techniques.

2.1 *Distributed Systems*

A distributed system in the most abstract sense is comprised of three elements: *nodes* the processes executing on servers, desktops, or sensors; *links* cable plant, air, or other physical transmission medium; and *protocols*. A protocol is a kind of agreement about the exchange of messages in a distributed system [49, 109, 250]. A complete protocol definition is very similar to a language description: it defines a strict syntactical format for valid messages; it defines data exchange procedure rules; and it defines semantics, a vocabulary of valid messages and their meaning [109]. The protocols grammar must be logically consistent and complete. The procedure rules should explicitly specify what is permitted or forbidden [109]. Finally, the sender and receiver must implement compatible rules for communication to succeed [109].

A distributed system can be abstracted by dividing it into *application stacks*, the components that make up the nodes; and *protocol stacks*, the layered architecture that implements the rules of communication. The application stacks are the operating systems and application software on any given node. To avoid combinatorial state explosion, protocols for distributed systems are often designed, developed, and implemented in layered architectures [49, 150, 252]. Each layer of the communications architecture implements services that are presented to the more abstract layers above it.

Figure 2.1 shows the Open Systems Interconnect (OSI) reference model proposed by International Organization for Standards (ISO) in 1982 [236].

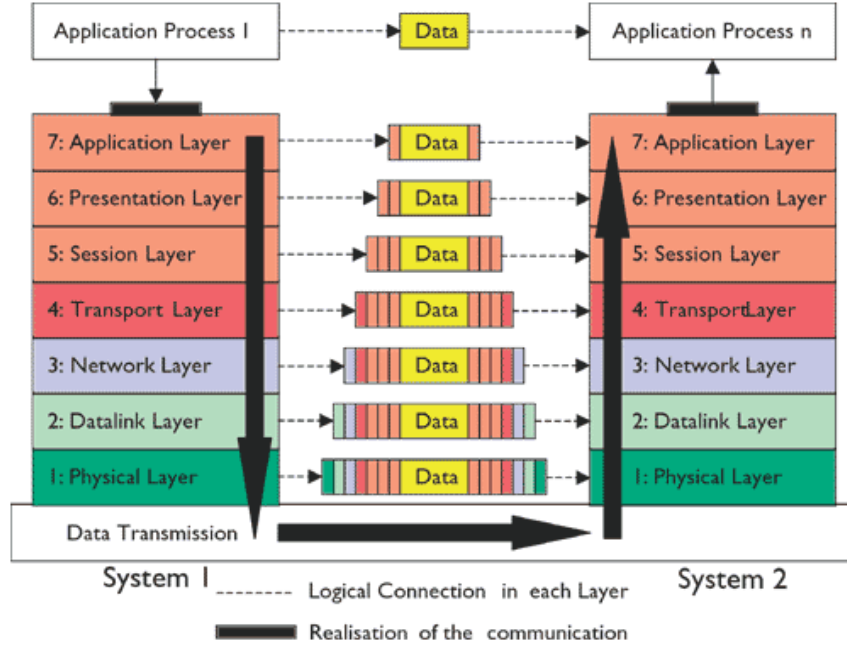


Figure 2.1: OSI Reference Model [236].

The protocol stack is the composite of the layers that are utilized by a distributed system. A protocol stack implements a protocol reference model. The Transmission Control Protocol (TCP) / Internet Protocol (IP) protocol family concentrates on the transport and network layers of the OSI reference model [49,243]. The protocol stack that supports a distributed system is completed by adding a data link and physical link implementation, such as Ethernet over fiber optic cable. Vulnerabilities in a protocol stack can be leveraged as a propagation vector for attacks on an application stack [12, Section 7.2]. Methods to exploit known vulnerabilities are readily available in pre-packaged frameworks such as Cain & Abel¹ [176] and Metasploit² [83].

To limit the scope of our research, we have selected to focus on application layer protocols and concentrate on the protocol stack over the application stack. Specifically, we will concentrate on application layer protocols that use the TCP/IP protocol family version 4 (IPv4) which defines much of the modern Internet [49,243]. We consider TCP/IP exchanges of TCP packets that encapsulate application layer protocols.

¹Cain & Abel - <http://www.oxid.it/cain.html>

²Metasploit - <http://www.metasploit.com>

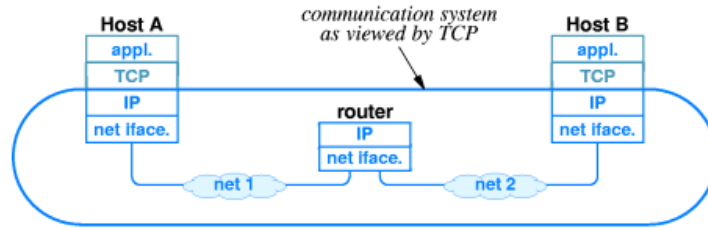


Figure 2.2: Internet from TCP Perspective [78].

Constraining the OSI reference model to just TCP connections leads to Figure 2.2. TCP, the dominant transport protocol on the Internet [119], uses the communications system at the layers below it as a simple black box and does not concern itself with the layers in the model above it.

Likewise distributed systems which implement application level protocols use TCP and the lower levels as a black box.

A Finite State Machine (FSM) is used to model any device that reacts to its environment and changes its state according to the inputs. The FSM model is often extended, to include outputs, as a Mealy-Machine. A well formed TCP packet has a source TCP/IP implementation (Host A) that uses an automaton to keep track of the state of a particular connection to a destination TCP automata (Host B) [33, 49, 78].

A client application communicates with a server application through a TCP client that connects to a TCP server through the network [33]. While a TCP connection is identified by Source Address/Port and Destination Address/Port pair; the temporal relationship is actually determined by the state of the TCP Server/TCP Client pair at the source and TCP Server/TCP Client pair at the destination. The distributed systems client application and server application also maintain separate state automata which transition states depending on the operators received. As shown in Figure 2.3, when a distributed system uses connection oriented TCP as a transport we are really dealing with at least four automata in each direction.



Figure 2.3: TCP Client/Server Topology [33].

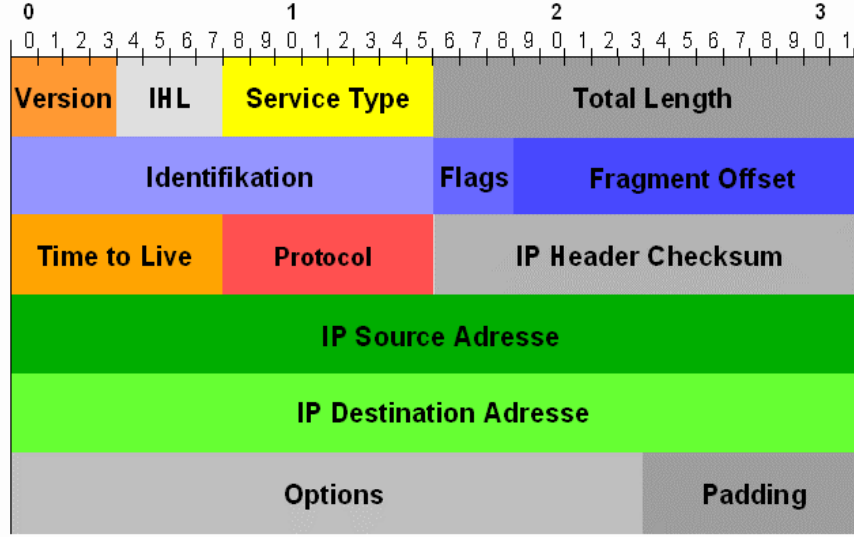


Figure 2.4: IP Packet Structure - 32-bit wide IPv4 IP packet structure [93].

Because we are considering application level protocols transported over IPv4 connections, this naturally gives rise to data structures embodied in protocol automata, formats of protocol operations, and data relationships from state transitions.

2.1.1 Data Structures. The three primary data structures in our selected problem domain are the IP packet structure, shown in Figure 2.4, the User Datagram Protocol (UDP) packet structure, shown in Figure 2.5, and TCP packet structure shown in Figure 2.6. Figure 2.4, Figure 2.5, and Figure 2.6 are laid out so they are 32-bits wide.

The UDP and TCP packets are encapsulated into IP packets at the network transport layer so their source and destination IP numbers are derived from the 32-bit IP Source Address and 32-bit IP Destination Address. The 16-bit wide source and destination ports are included in the packet structure (UDP or TCP) that makes up the IP payload [49].

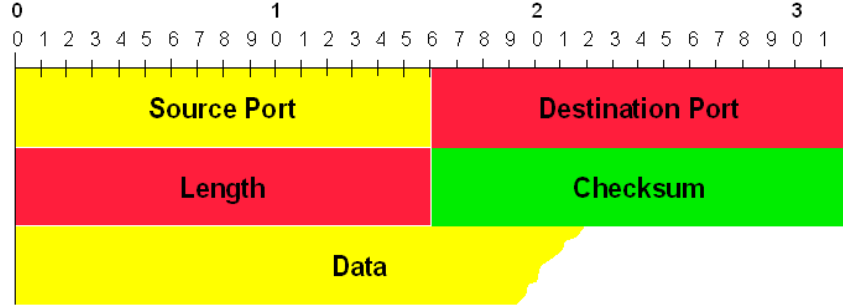


Figure 2.5: UDP Packet Structure - 32-bit wide IPv4 UDP packet structure [93].

The UDP packet structure is rather simple because the protocol does not provide connection oriented features. Distributed systems that use UDP for transport can be considered connectionless in the transport layer and must provide their own mechanism for re-transmission of failed communication [49]. Example uses for UDP are Domain Name Service (DNS) and games such as World of Warcraft.

The TCP packet structure requires more information to support reliable communications service. The TCP protocol provides for reliability, flow control, multiplexing, precedence, security, and connection oriented transfers [49, 243]

2.1.2 Data Relationships. Data relationships between packets occur at different levels of granularity: packet, connection or session. Another relationship is the temporal ordering of packet arrivals which can be disturbed by packet fragmentation. And a third is the possible causality of connection and session arrivals. Understanding these relationships is vital to choosing the parameters for clustering packets from traces into unique conversations between application level protocol endpoints.

2.1.2.1 Granularity. At the transport level TCP maintains a communication channel by using counters and synchronization flags. Application level protocols might also exhibit causal session structures. Communications at the application level also often have a logical session structure that associate packets and

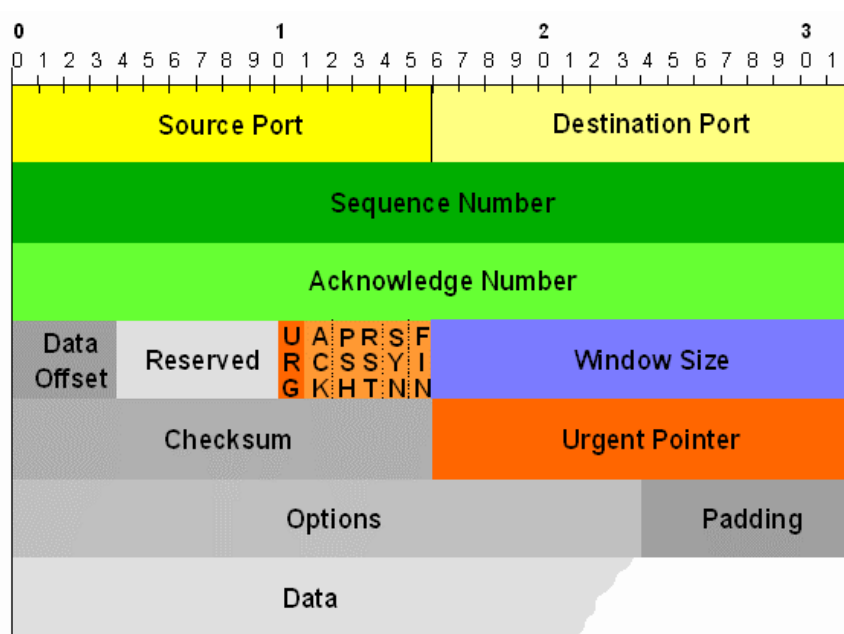


Figure 2.6: TCP Packet Structure - 32-bit wide IPv4 TCP packet structure [93].

connections. The level of granularity (packet, connection, or session) must be considered:

Packet Granularity - IP, UDP, and Internet Control Message Protocol (ICMP) [208] are all connectionless IPv4 protocols. That is, communications can be sent without prior arrangement. IP, ICMP, Address Resolution Protocol (ARP), Routing Information Protocol (RIP), and Open Shortest Path First (OSPF) routing protocol can be placed at the network layer (Figure 2.1). UDP, on the other hand, is a connectionless transport mechanism at the network transport layer (Figure 2.1).

Connection Granularity - At the transport level (Figure 2.1) the most fundamental relationship between the packet structures is a socket identified by the IP Source Address/Port pair and the associated IP Destination Address/Port pair. At an arbitrary point in time a TCP connection between a server and client can be identified by the IP Source Address/Port pair and the associated IP Destination Address/Port pair. Much TCP behavior is driven by timers and

timeouts. TCP inherently embodies greater temporal causality that is encoded in sequence and acknowledgment numbers in a specific TCP connection. This level of granularity, especially for TCP connections, has been widely studied (e.g. [42, 100, 119]).

Session Granularity - The IPv4 suite provides for sessions using Session Initiation Protocol (SIP) [221] which is a transport independent application layer control protocol. SIP supports session setup, maintenance, and teardown with one or more participants. SIP is used in voice, video, and instant messaging applications. Another alternative is International Telecommunications Union Standardization Sector (ITU-T) X.225 connection-oriented session protocol³ While session specifications are available application level protocols often use custom session level management.

2.1.2.2 Packet and Connection Fragmentation. Fragmentation is a feature of IP to support transport of packets across networks with different Maximum Transfer Units (MTU). Shannon [240] presents a detailed study of packet level fragmentation. The majority of fragmented traffic in the study was UDP. While ICMP, IP security (IPsec) [129], and Virtual Private Network (VPN) tunneled traffic were also commonly fragmented. Fragmentation occurred in 0.5 percent of the total traffic observed [240]. Reconstruction of connections from packet traces has the following issues [260]:

1. The IP datagrams may be fragmented.
2. IP fragments may arrive out of order.
3. Packets may be missing in the network trace because they were dropped during capture.
4. Adversaries might intentionally create non-deterministic situations with tools like fragrouter [1].

³ITU-T Recommendations are available at <http://www.itu.int/rec/T-REC/en>.

Connection oriented TCP also suffers from fragmentation effects when TCP segments are interleaved because the encapsulating IP fragments arrive out of order [20].

There is no direct algorithmic means for reconstructing a connections content from a trace of network packets. Correcting IP packet fragmentation and reassembling TCP connection stream requires a significant portion of the IPv4 protocol stack [260]. Turner [260] proposed the following possibilities: pre-process data with libnids which partially implements a Linux 2.0 TCP/IP protocol stack in user-space [275]; use packet reassembly code from Wireshark [281]; port a TCP/IP stack from open sources; or write a custom protocol stack from scratch.

2.1.2.3 Data Representation. Application layer protocols can be classified into binary and human readable ASCII text protocols. Text protocols restrict their operators and payload to printable ASCII text characters. Binary protocol operators are comprised of data fields that can be mapped to standard data types such as integers or strings. For example, SMTP uses an ASCII text representation for the operators while protocols like RPC and SMB/CIFS use binary representations. This means that Σ , the set of protocol operators, can be structured text or structured binary data.

2.1.3 Other Data Characteristics. User behavior also exhibits higher order structure. The problem is examined by Kannan [125] who defines a session as a group of network connections related to a network task. A *network task* is activity that emanates from an external event (the causal origin) [125]. We do not examine higher order session structures such as aggregate user session structures for web traffic.

Finally, protocols which use encryption (e.g. SSL, SSH, RDP) or are tunneled through encryption mechanisms offer additional challenges. Wright [277] presents an exploratory look at identification of packets in encrypted tunnels. Gebiski [91] also conducts an experimental evaluation of a model to detect and identify TCP protocols

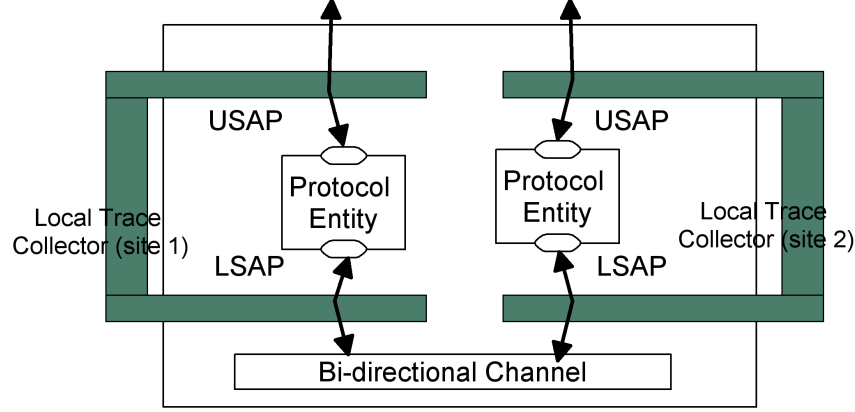


Figure 2.7: Trace Collection Architecture - proposed by Saleh [231].

in encrypted tunnels. We will not examine design recovery from encrypted or tunneled traces.

2.2 Network Trace Collection

Placement of the probe points for trace data collection must be considered, we should rationalize the observation points used. International Standard 9646 (IS-9646) defines four test architectures for OSI protocol conformance testing [68]. The four types are local, distributed, coordinated and remotes test architectures [257]. We can use the ISO test architecture descriptions to classify where a protocol reverse engineering effort collects trace data. The level of analysis can alternatively be classified according to the fidelity of observation described by Bhargavan in [24] which is partially determined by the placement of a monitor for a device under test.

Saleh also discusses placement of data collection points, shown in Figure 2.7 as the Upper Service Access Point (USAP) and Lower Service Access Point (LSAP) [230]. Like Saleh [230] we will refer to trace collections above the protocol under observation as Upper Service Access Points and trace collections below the protocol under observation as Lower Service Access Points. Saleh considers data collected at the LSAP to provide protocol primitives and data collected at an USAP to provide service primitives to layers above.

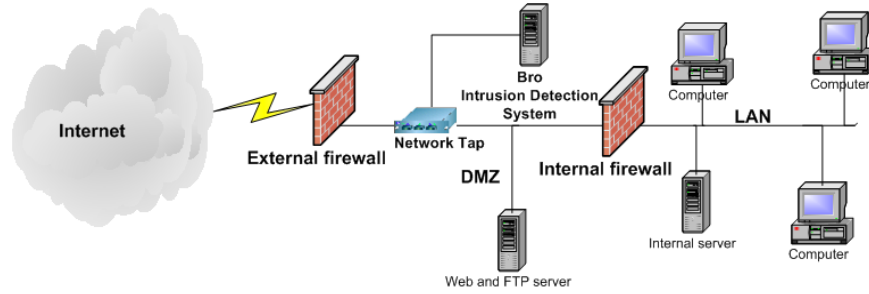


Figure 2.8: Bro deployment with network tap - the network tap duplicates the physical layer signaling for in-line full-duplex traffic analysis by the Bro Intrusion Detection System [141].

In a TCP/IP network *port mirroring* and *hubbing out* are two techniques for trace collection that do not require instrumenting the protocol under investigation. Port mirroring, or port spanning, is supported by some Layer-3 switching devices. The switching device copies all traffic on a user specified port to another physical port on the switch [233]. Hubbing out is a technique in which a target device and analyzer are located on the same Ethernet network segment by plugging them in to the same hub [233].

Packets dropped from the sample data by the capture mechanism will be an issue for high bandwidth traffic. Solutions include using custom high-speed collection hardware, such as [200], implementing hardware to duplicate network traffic at the physical layer, or limiting the study to low-speed protocols that can be captured with a high degree of confidence. Bhargavan used a modified Linux system to sniff TCP network traffic by hubbing out on a 100 Mbps connection, describing this configuration as a co-networked monitor [24]. The implementers of the Bro intrusion detection system recommend the use of a physical layer network tap, shown in Figure 2.8, that exactly duplicates the physical layer signaling for in-line full-duplex traffic analysis [141].

2.3 Application Level Protocol Data Flow Recovery

We differentiate data flows from the usage of network flows that is commonly used in graph theory [65]. This is because we are not interested in modeling the network as a graph but instead modeling the actual flow of data, thus data flow, which is captured in a trace file. Others have characterized packet data flows as: *packet trains* [118]; *streams* and *torrents* [35]; or *flights* [239]. As discussed in Section 2.1.2.2, we can define a data flow as a quintuple $\langle \text{source IP, source port, destination IP, destination port, protocol} \rangle$ [264]. An advantage of using tuples is that they support formal definitions of the operators that apply. Also, defining the mathematical symbology allows us to discuss the problem domain in a more compact form.

2.3.1 Naïve Flow Membership. In the case of TCP/IP, at the packet level of granularity, we can use the TCP setup (three-way handshake) to recognize the beginning of a partial flow and TCP teardown to recognize the end of a partial flow. Several studies have used TCP flows defined by the SYN/FIN control mechanism in TCP to denote flows [42]. This representation is adequate if we consider network traffic as bi-directional flows comprised of arbitrary groupings of packets defined by the attributes of their endpoints. Our naïve definition of flow membership does not address the temporal nature of application protocol communications at the session level. Also, it does not account for timeout or connection loss.

2.3.2 Accurate Flow Membership. In reality, we will not get enough information from the five-tuple $\langle \text{source IP, source port, destination IP, destination port, protocol} \rangle$ to accurately determine a packets flow membership. The traditional use of well known port numbers⁴ to identify application level protocols is in question. In contemporary network traces port numbers no longer provide reliable recognition of application protocol traffic types. Current Internet traffic often ignores established port number conventions [154]. Statistical techniques have been proposed to accu-

⁴The Internet Assigned Numbers Authority Port Numbers provides a list of well known ports [116].

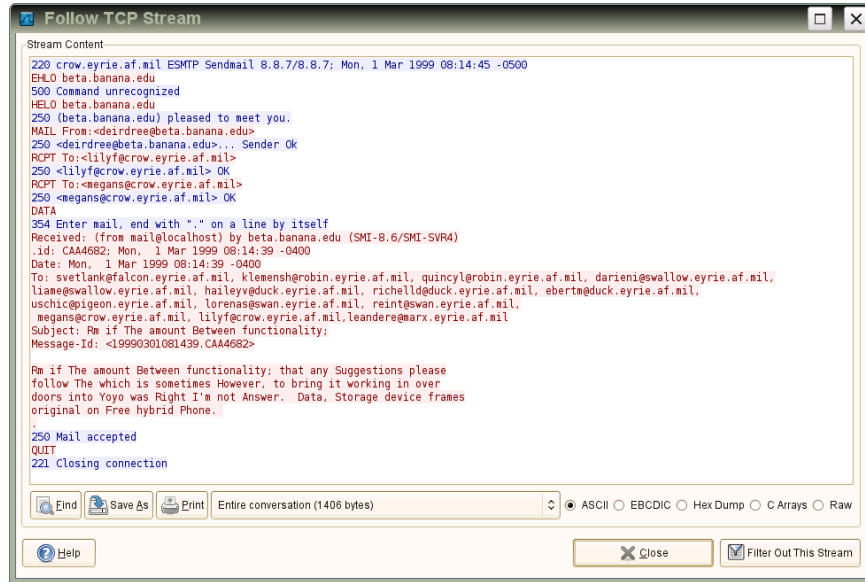


Figure 2.9: Wireshark Following SMTP Conversation - Wireshark can reconstruct the conversation between the server and client using internal knowledge of the SMTP protocol.

rately identify the type of application protocol encapsulated in connections. Ma [154] presents a Markov process model technique and a longest common subsequence approach. Both techniques provide statistical recognition of the application level contents encapsulated in transport level connections.

Figure 2.9 shows an application level SMTP data flow reconstructed by following the underlying transport level TCP connection using Wireshark. Wireshark can reconstruct the conversation between the server and client because it uses internal knowledge of the structure of the SMTP protocol.

2.3.3 Stateful vs. Stateless. Application data flow membership will also be impacted by the state characteristics of the protocol under consideration. A stateful server maintains persistent information about its clients while a stateless server does not. SMTP and POP3 are stateful while HTTP is stateless. State information is added to web applications that use HTTP by the use of cookies that encode the session id and/or session state. If the server is stateless but maintains *soft-state*, data that is maintained for the client on the server for a limited time [253, p.91], then we

must have a method that detects early session termination or incomplete sessions. As is the case when a user navigates away from an HTTP based web application without actively terminating the session. This is difficult without a construct that handles timeout events.

Paxson and Floyd’s study of wide-area TCP arrival processes found that session arrivals were well modeled by Poisson processes with hourly rates even though individual connection arrivals were not [195]. Nuzman [186] found that the arrival of HTTP connections aggregated into sessions also reflect a Poisson process. Kannan [125] used this observation as a key part to discovering and characterizing causality in network traffic. Meent [264] uses a 20 second interval for membership in a flow at the packet level. This means TCP/IP packets identified by the quintuple must be within 20 seconds of each other to be classified as members of the same flow. McGregor also provides some clustering techniques for classifying flows in [165].

2.3.4 Single-connection vs. Multi-connection. Data flow membership will also be impacted by how the protocol uses the underlying transport. SMTP and POP3 session boundaries are easily detected because each session is encapsulated in a single TCP connection [112,113]. This means the connection and session granularity are equivalent for SMTP and POP3. On the other hand, version 1.1 of the HTTP specification allows for re-use of open TCP connections for multiple requests [80]. HTTP based web applications use session identifiers to associate sessionless HTTP requests into a logical application session.

To determine the start and end of a complete flow or session of an application level protocol we must understand the operators that support session setup and tear-down. Another possibility is to detect session identifiers encoded in the packet traces of the application layer protocol under consideration.

2.3.5 Single-channel vs. Multi-channel. Another concern is the number of connection level channels that make up the communication. Session detection for

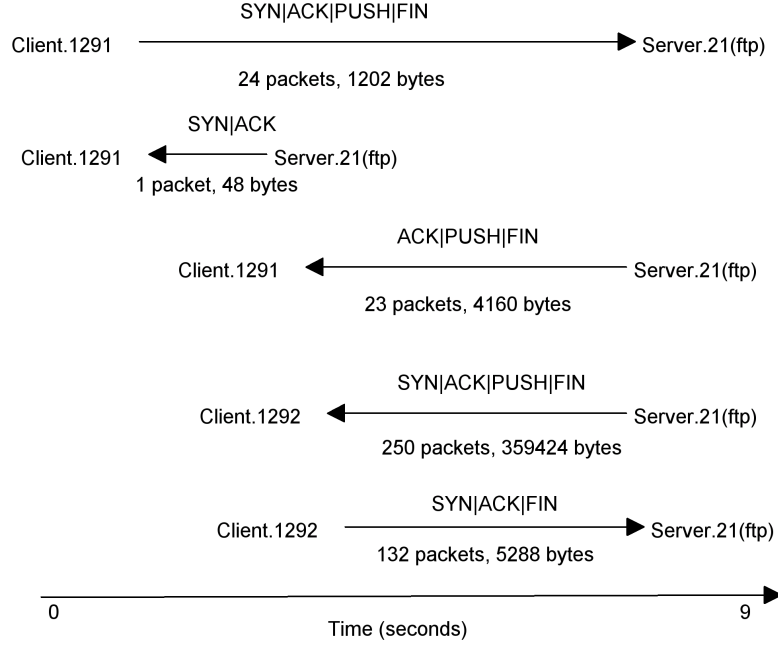


Figure 2.10: Flow level breakdown of a simple FTP transfer - the control channel is on port 1291 and the out-of-band data channel on port 1292 [15].

single-connection, single-channel protocols, like SMTP or POP3, can be determined from TCP socket connection status. For more complex multi-channel protocols we must understand the internal structure of the protocols (e.g. FTP and RPC) to properly group the packets and connections that make up the application level data flows [15, 260]. A notional multi-channel FTP flow is shown in Figure 2.10 which shows the control channel and an out-of-band data channel.

2.4 Application Level Network Traces into Automata

We must address the following four issues:

- Network trace collection.
- Application level protocol data flow recovery.
- Protocol format (Σ) recovery.
- Protocol transition function (δ) recovery.

We discussed the issues of trace collection in Section 2.2 and data flow recovery in Section 2.3. Here we concentrate on format (Σ) recovery and transition function (δ) recovery from existing traces.

Protocol format recognition is coupled with the data portion of a protocol. As discussed by Lee in [142] the data portion specifies functions that involve parameter values associated with messages.

Protocol transition function recognition focuses on the control portion of a protocol over the data portion [142]. Given two endpoints of a distributed communications system, a sender $S = \langle Q_s, \Sigma, \delta_s, q_{s0}, F_s \rangle$ and receiver $R = \langle Q_r, \Sigma, \delta_r, q_{r0}, F_r \rangle$, how can we go about recovering a model of δ_s and δ_r ? What is the minimum knowledge we need *a priori* to recover the design using only a captured data flow of the finite members of Σ ?

While, we can recover the expected Q, Σ, δ, q_0 , and F from protocol specifications, by analysis of source code, or even reverse engineering binary code, what if specifications or source code are not available?

2.5 Protocol Design Recovery

We must determine operator formats (data portion) and determine automata (control portion). This means we must know or infer the states, operators, transitions, initial state(s), and final state(s) which define the behavior of the protocol under consideration. To frame the discussion we present an overview of forward engineering and re-engineering practices. Figure 2.11 shows the inter-relationship of re-engineering practices.

2.5.1 Forward Engineering. Forward engineering is the traditional engineering process of moving from high-level abstractions to the physical implementation of a system [40, 214]. Protocol engineering is an interdisciplinary approach which emphasizes the use of sound engineering principles and formal methods to develop reliable communication software [230]. Protocol forward engineering efforts can utilize formal

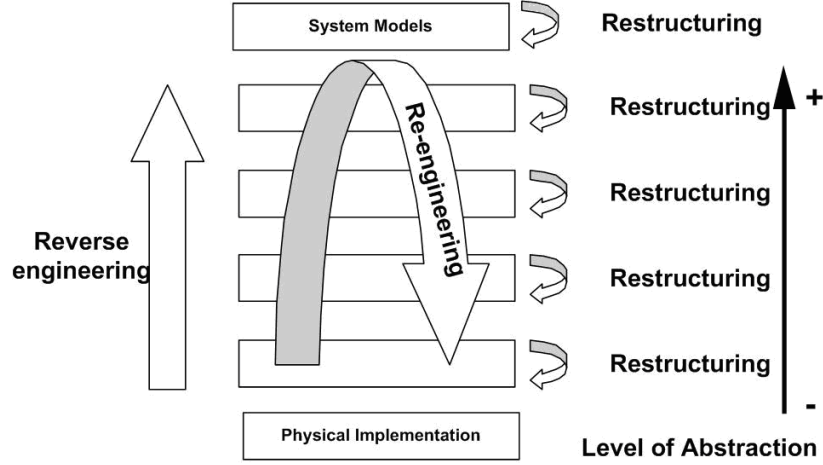


Figure 2.11: Relationship of Re-engineering Practices - Reverse engineering attempts to recover higher levels of abstraction, restructuring modifies system artifacts at the same level of abstraction and re-engineering uses reverse engineered artifacts to generate a new physical implementation [214].

techniques for analysis and modeling but many protocols are designed and implemented using informal approaches [109, 231]. Even widely used protocols like HTTP, SMTP, and POP3 rely on English language descriptions of correct control flow.

2.5.2 Reverse Engineering. Reverse engineering is the practice of discovering the technological principles of a device/object or system through analysis of its structure, function, and operation [40, 108, 214]. Software reverse engineering concentrates on analysis of software through disassembly and debugging of a software program. In some sense the inverse of forward engineering software reverse engineering is also referred to as *reverse code engineering* (RCE).

2.5.3 Protocol Reverse Engineering. Protocol reverse engineering is the application of reverse engineering, often including RCE, to recover the automata and operators which define the protocol. Protocol reverse engineering can be tightly coupled with RCE. If the protocol specification is not available, RCE of source code can provide many clues to the structure of the protocol. RCE can also be applied to the binary programs that implement a protocol to approximate the original design/engi-

neering decisions. Protocol reverse engineering without access to specifications or source code can be significantly more challenging.

2.5.3.1 Static vs. Dynamic. Static vs. Dynamic protocol reverse engineering (see Saleh [230]) are differentiated by information source. Static protocol reverse engineering is the recovery of protocol information from specification documents and implementation artifacts including system documentation, source code, or even reverse engineered binary code. The process is static from the perspective of the protocol under inspection. RCE of binary code will likely involve dynamic runtime analysis at the local level. Lie proposed creating models from protocol code using an extensible compiler system and applied the system to analyze cache coherence protocols in multi-processor systems [149].

Dynamic protocol reverse engineering is the recovery of protocol information from observations of the system in action. If the protocol is part of a layered architecture (such as the TCP/IP protocol suite which implements in part the OSI reference model) the traces may be collected at an observation point established at a layer below the protocol or a layer above the protocol. An observation point above the protocols layer should collect events that are caused by service requests from layers above the protocol under observation.

Much like putting a multi-meter or oscilloscope at the inputs (lower service access point) and outputs (upper service access point) of an electronic circuit to determine its internal operation by collecting traces of inputs and outputs we will collect packets from the lower service access point (LSAP) and upper service access point (USAP) of the protocol automaton under consideration. Then we can attempt to infer the internal operation of the protocol automaton or construct an equivalent automaton.

2.5.3.2 Who needs it? Distributed systems rely on the correctness of both open and proprietary protocols to provide functionality. As an example,

Microsoft’s proprietary Server Message Block/Common Internet File System (SMB/-CIFS) is often encapsulated in TCP connections [105]. Deep understanding of the protocols which underpin distributed systems is increasingly important to computer security efforts [100, 135, 250]. Protocol reverse engineering is often the only option available to develop an understanding of proprietary protocols that allows us to validate that the protocol implementation is correct, reliable and secure.

Protocol reverse engineering has been proposed for a range of purposes including: conformance testing [143]; design recovery [231]; to develop interoperable interfaces between incompatible protocols [194, 248]; as a means to enhance network security analysis [100]; and to develop signatures for network intrusion detection systems [178]. Despite these practical uses the practice is hindered by legal obstacles designed to thwart theft of trade secrets and, a perhaps deserved, perceived lack of legitimacy [111, 199].

2.6 State-of-practice

State-of-practice reverse engineering tools and techniques for file formats and binary executables are presented through several hacker oriented books and websites (e.g. [72, 75, 87, 98, 132]). Collection and observation of TCP/IP network traffic is also well covered (e.g. [48, 84, 98, 190, 242, 251, 279]). There are a few studies (e.g. [82, 100, 143, 229]) and web articles [210] that discuss protocol reverse engineering. The topic, possibly due to its somewhat underground nature, has not received broad academic treatment.

Fortunately, the state-of-practice is documented by open source projects that apply protocol reverse engineering to re-implement proprietary protocols. Three selected open source projects that use protocol reverse engineering methods are Pidgin, Samba, and Rdesktop [38, 50, 258]. Because the protocol specifications are not completely available all three projects rely on reverse engineering.

2.6.1 Tools. The basic toolset capabilities required for TCP/IP protocol reverse engineering is the ability to capture/record, manipulate, and analyze TCP/IP packets. There is a wide range of open source, public domain, and commercial tools that provide capabilities that are useful for protocol reverse engineering efforts ⁵. Sonnenburg et al [245] argue that open source machine learning is key to supporting experimental reproducibility. Joyner and Stein also argue the value of open source software to mathematical studies in an opinion piece [122]. In the interests of reproducibility we examined tools that are openly available either as open source or public domain. Below we discuss several open source tools that can assist with TCP/IP protocol reverse engineering within the context of our four reverse engineering problems: trace collection, data flow recovery, format recovery, and transition function recovery

2.6.1.1 Trace Collection. One commonly referred-to tool is Wireshark shown in Figure 2.12 [48]. Wireshark, formerly called Ethereal, is an open source packet capture program. It includes dissector algorithms which recognize and parse many text and binary application level protocols [190, p.79]. Another often-mentioned tool for packet capture is tcpdump [51]. Packet data flows are stored in several file formats. Because Wireshark and tcpdump both use pcap, libpcap under UNIX and WinPcap library under Windows, their file formats are compatible. WinPcap on Windows and libpcap on UNIX provide similar application programming interfaces (API) and are commonly referred to as the pcap API. The tcpdump website that hosts libpcap provides links to several related utilities and projects built with the pcap API.

In the assessment of Kreibich, author of NetDude, the trifecta of Pcap, tcpdump, and Wireshark form the *de facto* standard tool set for TCP/IP protocol reverse engineering [133].

⁵List of tools (e.g. [56] and [51, Related tools]) are readily available via the Internet.

2.6.1.2 Data Flow Recovery. There are tools that can reconstruct connection level traces from packet traces. For example, NetDude [133] includes a demux (de-multiplexer) plugin that breaks packet traces into sub-traces along the TCP connection boundaries. The sub-traces are stored in pcap compatible trace files. Regrettably, the sub-traces do not contain the complete communication between the server and client. Bhargavan proposes the development of Network Event Recognition Language (NERL) [26, 27] to reconstruct application level protocols for monitoring purposes. A NERL implementation is described in [25], including analysis of SMTP, but a reference implementation is not made publicly available.

There are some tools to reconstruct session level traces from packet traces. Chaosreader [34] and tcpflow [73] both recreate session flows from packet traces. Unfortunately, the output from both tools is formatted for human review not automated processing.

2.6.1.3 Format and Transition Recovery. We discovered no publicly available tools that provide automated format or transition function recovery from network traces. This is further discussed in Section 2.8

2.6.1.4 Miscellaneous Tools. The pcap API is used by a wide range of utilities and tools to generate statistics from packet traces. Automated support for packet level analysis is available from Tstat [223] and tcptrace [191]. Several tools add features to support online and live packet manipulation and replay of captured data. Flowreplay [260], based on Turner’s earlier work on tcpreplay for datalink replay, was designed to replay TCP/IP traffic at the transport and application layer. Scapy is an interactive packet manipulation program [29]. Paxson created tpanaly [195] to automatically analyze TCP implementations at the packet level of granularity. The tool is not publicly available. Other tools are focused on filtering traffic from real time collections for direct observation by a human operator. Examples include: Trafshow [272], Ngrep [217] and Ntop [64].

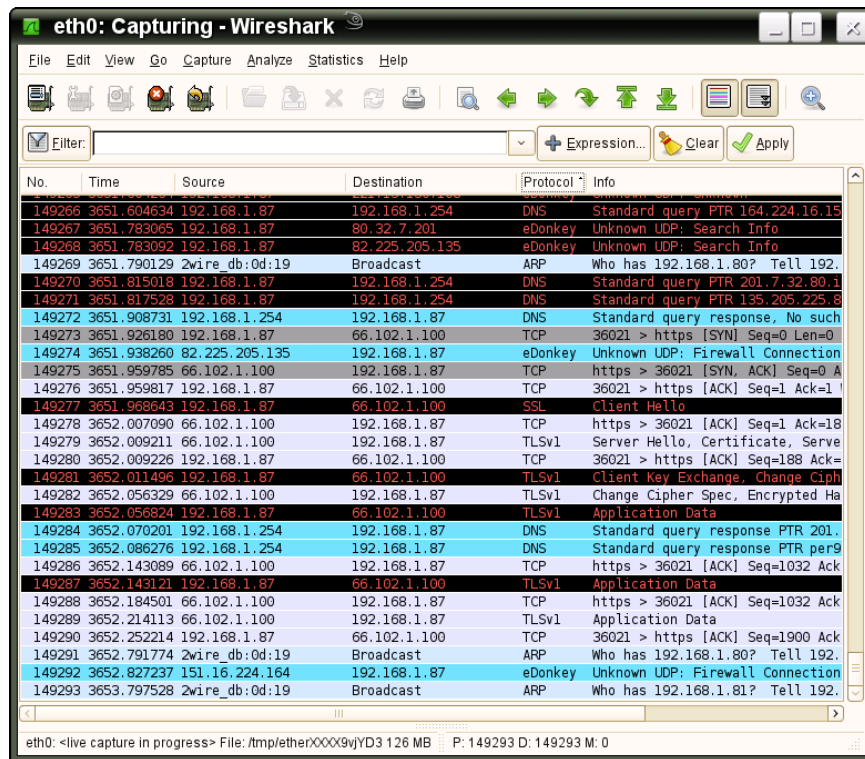


Figure 2.12: Wireshark - Ready for “test, capture, and stare”.

2.6.1.5 Programming Toolkits. Finally, there are several toolkits, besides Pcap, which provide APIs to ease the development of network tools. The libnet library [234] is a toolkit allowing the construction and injection of packets. Another toolkit is libdnet which supports low level network operations for several languages (C, C++, Python, Perl and Ruby) on many UNIX variants and Microsoft Windows [244]. The libevent toolkit provides an event oriented API. The event abstraction allows developers a design alternative to polling loops and threads when processing network traces [160]. Libevent supports per event timers with callbacks on timeout. Another library is libnids which provides a Linux version 2.0 TCP/IP protocol stack in user space [275]. The libnids toolkit uses libpcap and libnet internally to provide IP fragment reassembly and IP stream reconstruction [275]. The libnet, libdnet, libevent, and libnids toolkits are accessible from scripting languages allowing rapid adhoc prototyping.

2.6.2 Techniques. Although protocol reverse engineering involves tools, the key is the reversers ability to understand the assumptions and design decisions of the people who created the protocol specification or implemented the protocol, and then undermine them. Reverse engineering requires in-depth knowledge of myriad technical specifications and specific implementations coupled with an understanding of the engineering decisions of the original designers.



It should be noted that the projects used both online and offline techniques.

One protocol reverse engineering technique used by all three projects contributors is described, somewhat lightheartedly, as test, capture, and stare [258]. The technique is an adhoc approach that depends on fast turnaround of simple tests that involve capturing network traffic resulting from varying parameters in operators. Test, capture, and stare informally defines the state-of-practice for protocol model reverse engineering from captured data flows.

Two other prominent techniques are protocol filters and protocol specific scanners. A protocol filter is a Man in the Middle (MITM) proxy server that can make changes to protocol data before it is passed on to a target server. MITM proxy servers, as shown in Figure 2.13, use session hijacking techniques such as ARP poisoning [98, p.215] or DNS poisoning [98, p.216] to re-direct traffic from the original communication between a client and server on network path A. When the MITM proxy server is active the client communicates with the MITM proxy on path B and the server communicates with the MITM proxy on network path C. Filters in the MITM proxy allow specific or global substitutions of packet contents. A MITM proxy server based protocol filter can be essential in online (or live) reverse engineering [82, p.9].

A protocol scanner uses known signpost values, such as error codes, to guide exploration of protocol structure. Scanners can be used to find new parts of a protocol or to determine some properties of a protocol operation [258]. One challenge of

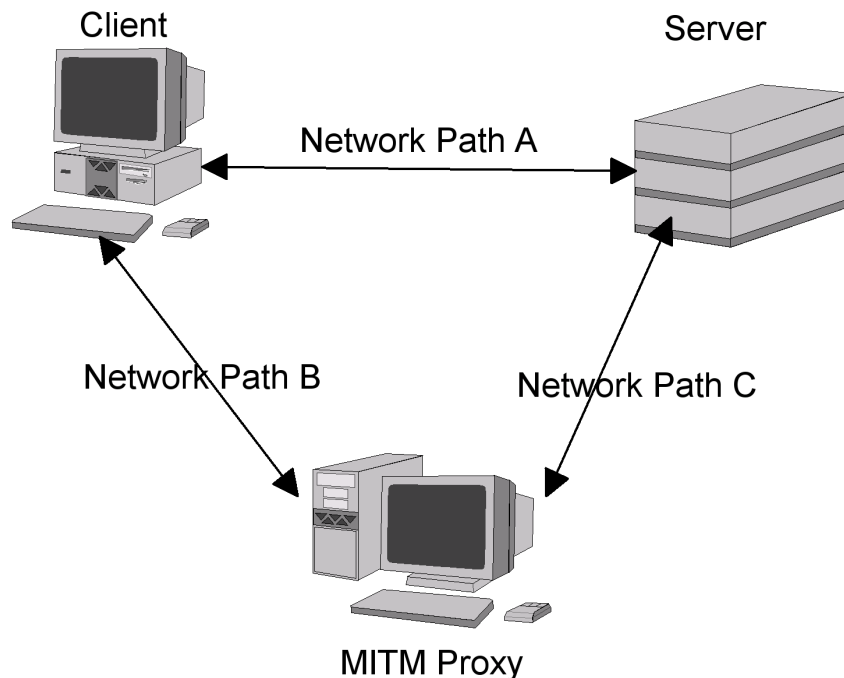


Figure 2.13: MITM Attack Topology [82, Figure 2.1].

developing protocol scanners is recognizing the exact meaning of known signpost values and how the proprietary parser responds to those values [258].

2.7 Case Studies

Here we present an overview of protocol reverse engineering as used in three selected open source projects that rely on protocol reverse engineering methods: Pidgin [50], Samba [258], and Rdesktop [38]. While there are other projects that apply protocol reverse engineering we find that the selected projects are significant because they are concentrated community driven efforts that openly present the tools and techniques used. In fact, the results of each project are available as open source.

2.7.1 Pidgin. Pidgin is a multi-protocol Instant Messaging (IM) client that supports the use of proprietary IM servers [50]. Pidgin's plug-in architecture allows independent efforts towards reverse engineering and re-implementing proprietary IM protocols. It is difficult to characterize the overall protocol reverse engineering ap-

proach used by Pidgin contributors. In part, because IM protocol specific plug-ins are developed independently. Although protocol filters are available for IM protocols it is not clear if they were used by plug-in authors [2]. In an E-mail conversation the implementer of one protocol mentioned that Wireshark was critical. While the client fully implements open IM standards such as Extensible Messaging and Presence Protocol (XMPP) there are still communications issues between the proprietary IM clients and proprietary servers. One example is that file transfers using the Windows Live Messenger compatible ⁶ .NET protocol have not yet implemented faster peer-to-peer functionality [241]. Pidgin plug-ins for proprietary IM protocols, such as Windows Live and Yahoo, must be patched when the protocol is changed. The delay between protocol changes and working client software caused by reverse engineering can be months.

2.7.2 Samba. Samba is an open source implementation of the closed source proprietary Microsoft Windows SMB/CIFS implementation. The Samba project, unlike Pidgin, must support several different session and application level protocols to implement SMB/CIFS functionality. The project has taken over 12 years to manually reverse engineer SMB/CIFS [58, 258]. The effort has successfully achieved interoperability with Microsoft Windows file and print sharing services. It has also implemented Windows NT domain controller services and the project plans to implement active directory capabilities [105, 258]. Samba is the basis of Microsoft Windows network interoperability for many UNIX based systems including Apples Mac OS X since version 10.1 (Shown in Figure 2.14) [63]. Recently, the project established the Protocol Freedom Information Foundation (PFIF) to acquire protocol documentation Microsoft made available due to European Union court decisions [259].

Samba is unique in that the effort has been guided by consistent leadership. Andrew Tridgell the initiator of the project, much like the Linus Torvalds for the Linux kernel or Richard Stallman for GNU projects, serves as the public representative of

⁶Windows Live Messenger – <http://get.live.com/messenger/overview>



Figure 2.14: SAMBA on Macintosh OS X 10 [63].

Samba. Tridgell proposed *The French Café Technique*, also called network analysis, as the projects overarching approach to reversing the SMB/CIFS protocol family [258]. Samba contributors use a range of techniques and tools beyond test, capture and stare to support their reverse engineering efforts. Samba contributors developed a SMB protocol scanner called trans2. It includes dozens of sub-commands, what we term operators, which implement various types of file and file system queries. The scanner tries different information levels, data sizes, and object types to determine what operations exist and what sizes of data are involved [258]. The project also documents informal protocol reverse engineering techniques that are not detailed in the other projects. Two of the techniques are trial and error analysis and dual server and backtracking

2.7.2.1 Trial and Error Analysis. Complex protocols tend to have many error values [258]. To determine what each error code means Tridgell recommends writing an error driven protocol scanner [258]. Tridgell also recommends an error mapping approach. As an example, when targeting a file sharing protocol he recommends modifying the server to return error XXX for filename 'test_XXX.dat', then asking the proprietary client to access filenames from test_001.dat to test_999.dat. Finally, the message returned to the client must be collected for analysis. Proxy error mapping is a technique invented by Andrew Bartlett to discover the correct mapping

between Microsoft Disk Operating System (MS-DOS) error codes and Windows NT status codes [258]. Proxy error mapping extends trial and error analysis by inserting a protocol proxy between the server and client. The proxy contains error codes that were known from MS-DOS clients (signpost values), performed the same operations that returned the MS-DOS error codes against reference Windows NT servers. Proxy error mapping works in this case due to *a priori* knowledge of the MS-DOS error codes.

2.7.2.2 Dual Server and Backtracking. The dual server technique can be used to fine tune understanding of a protocol. The basic concept is to write a client that connects to a reference server in parallel with the reverse engineered server, then to systematically generate protocol operations and finally compare the results from both servers [258]. One problem with this technique is the protocol may have temporal dependencies between current and past operations. It is possible that both servers will process several, maybe even thousands, of operations before generating an error condition. For this reason the operators sent to the dual servers should be recorded for replay. Suspected erroneous operations can be removed from the replay and flagged for further consideration if the error condition is not returned by the reference server. The process of removing suspect operators and replaying is also referred to as differential analysis [258].

2.7.3 Rdesktop. The rdesktop project, initiated by Matthew Chapman, implemented an open source client for Windows NT Terminal Server and Windows 2000/2003 Terminal Services [38]. It supports the Microsoft Remote Desktop Protocol (RDP) on several UNIX based platforms with the X Window System (See Figure 2.15). The RDP protocol is an extension of International Telecommunications Union Standardization Sector (ITU-T) T.128 multipoint application sharing protocol [38, 170]. This project, unlike Pidgin and Samba, was focused on a single protocol. Even though Microsoft's proprietary implementation of the RDP protocol is partially specified in ITU-T T.128 the project required significant effort because the protocol traffic is en-

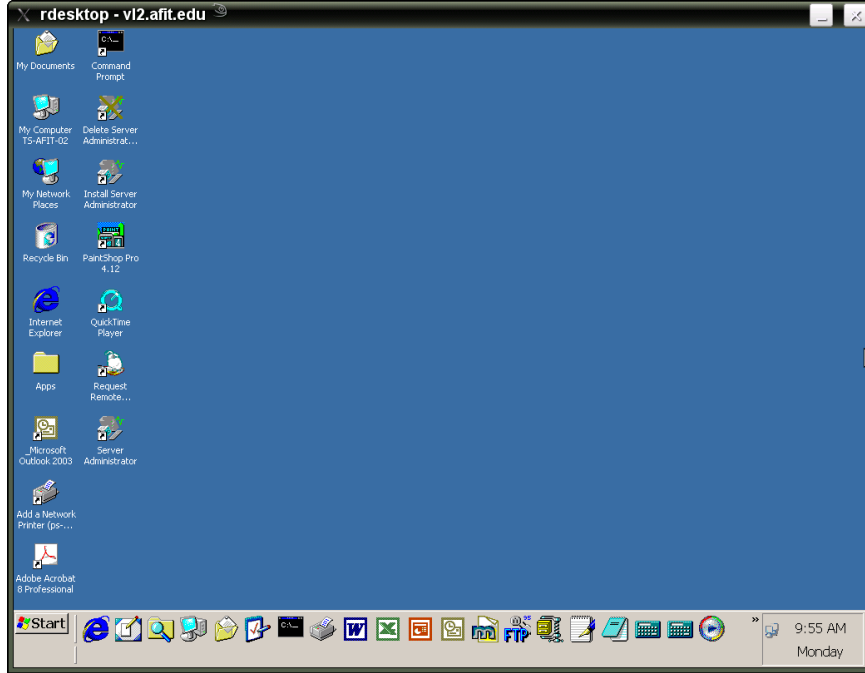


Figure 2.15: Rdesktop Connection.

encrypted. Like Samba the project utilized the test, capture, and stare approach to incrementally build a working RDP implementation [52,103]. Also, like Samba, project contributors used protocol filters to conduct MITM attacks that revealed specifics of Microsofts RDP protocol [82]. The project implemented *rdpproxy*, an RDP specific protocol filter, to perform advanced processing of RDP data flows [82].

2.8 *State-of-Art*

The contributors to the projects discussed in Section 2.7 were highly dependent on variations of test, capture, and stare. While the Samba project presents additional adhoc methods such as specialized protocol scanners, proxy servers and partial implementations of reference servers each method depends on the knowledge incrementally encoded in their re-implementation of SMB/CIFS. Even after multi-year efforts, the state-of-practice for protocol reverse engineering has not advanced beyond painstaking, time-intensive, manual scrutiny of packet captures using tools like *tcpdump* and *Wireshark*.

As discussed in Sections 2.1.1 and 2.1.2, the key data structures and data relationships we must understand are the protocol automata and operator formats. At its heart, protocol reverse engineering involves inferring the packet format of operators and the structure of the protocol automata. The most obvious approach is of course to access the protocol specification if it is available. Another alternative is to review the source code for the protocol implementation. Unfortunately, many protocols implementations do not release specifications or source code for public review.

While it is possible to apply RCE techniques to recover an approximation of the protocol source design we choose not to examine this alternative, instead concentrating on model recovery from network traces. Regardless, we acknowledge that a robust protocol reverse engineering effort has much to gain from RCE of the programs that implement the protocol under consideration. In fact, RCE of source code or binary code may be the only option if a specification or other documentation is not available.

Furthermore it should be kept in mind that recovery of operators or automata is limited by the completeness of the captured data flows. If the captured data does not include a complete sample of the operators used by the protocol we will not be able to completely construct all the operators or an accurate automaton. Lets examine current research efforts for protocol format recovery and protocol automata recovery separately.

2.8.1 Protocol Format Recovery. Building a dictionary of the operators an application protocol implements requires us to understand the format of the payload data encapsulated in TCP traffic.

An approach drawing from bio-informatics research is proposed by the PROTOS protocol genome project [104]. The project intends to utilize automated structure inference techniques for the purpose of developing automated testing tools. To date the project has provided only notional results.

A similar concept that crops up when searching on protocol structure inference is a concept called Protocol Informatics. Protocol Informatics was introduced by a

security analyst named Marshall Beddoe in 2004. The only remaining evidence of the effort is a Python⁷ implementation of some of the concepts available on the Internet at [18].

Another initiative called Discoverer, from Microsoft Research, uses machine learning techniques including clustering to infer protocol packet format idioms [58]. The authors evaluated their approach over HTTP, Remote Procedure Call (RPC) and SMB/CIFS. The authors focused on the correctness, conciseness and coverage of their format inference leaving automata inference for future work. The inferred packet formats covered over 95 percent of their captured data flow traffic [58]. Unfortunately, the algorithms and data sets used in their analysis have not been released to the public.

Borisov et al describe their Generic Application-Level Protocol Analyzer (GAPA) in [32]. The program implements a protocol specification language (GAPAL) using a syntax format similar to Augmented BNF. GAPAL is used to prototype application protocols and supports modeling of the underlying protocol state machine. While the authors mention that the tool can potentially enable the automatic generation of vulnerability signatures they do not implement any automated inference of the underlying protocol format or protocol automata. Also, the GAPA implementation and GAPAL specification are not publicly available.

Recently, Fisher et al propose automated inference from ad hoc data to generate PADS data description language [81]. A generic structure discovery algorithm is presented in Pseudo-ML in [81, Figure 5]. While source for the inference algorithm is not publicly released an implementation is available through the PADS project web site⁸.

2.8.2 Recovering Automata. Automated recovery of protocol models as different types of automaton has been proposed in various forms through out the last decade. Message Sequence Charts and Communicating Finite State Machines are two

⁷Python Programming Language – <http://www.python.org/>

⁸PADS project – <http://www.padsproj.org>

representations that have been proposed for model recovery by automata synthesis and automata inference from protocol execution traces. Synthesis differs from inference in that synthesis uses a complete sample to construct the target automaton. An inference procedure might not have a complete, or even characteristic, sample to generate hypothesis automaton.

2.8.2.1 Message Sequence Charts. The use of Message Sequence Charts (MSC) for communications systems is discussed in detail in ITU-T Z.120 [117]. Alur provides an algorithm for synthesis of MSC and establishes the foundational theory for MSC inference. In Design Recovery from Observations [261]. Ural et al propose recovery of protocol designs by analysis to build MSC. Their approach recovers a lattice of repetitive sub-functions from a series of observations [121]. After recovery the lattice is manipulated to synthesize an MSC model. While Ural et al implemented the synthesis algorithm in C++ they did not implement a trace collection or processing architecture, instead using generated text files as input to their system [261, Section 4]. If we choose MSC to model recovered protocol designs then the algorithms presented could serve as an analytical backend for protocol performance properties. The source code and executables to their implementation are not available to the public.

Another effort that partially solves the problem of protocol model recovery is *Synthesizing Models by Learning from Examples* (Smyle)⁹. MSC inference is applied to conformance testing by [31] in the Smyle system. MSC are used as inputs to the model synthesis system. Smyle uses inference learning from MSC to develop a message-passing automata (MPA) model [31, Definition 3]. Smyle can synthesize a model from a given labeled scenario (MSC samples marked as positive or negative). The system uses an extension of Angluin’s L^* algorithm to support MSC using a LearnLib (See Section B.3.1) based inference mechanism [31, Section 4]. Unfortunately, Smyle requires manual creation and labeling of the positive and negative samples. Like Ural et al, Smyle does not incorporate a trace collection or processing architecture

⁹Smyle - <http://smyle.in.tum.de/>

but could also serve as analytical backend to check performance properties. Smyle executables are available for research purposes by request. Source code is not made available at this time due to third party involvement.

Finally, the company Event Helix provides a modified version of Wireshark to synthesize MSC like graphs from packet traffic at the packet level of granularity [77]. The product does not support synthesis or inference of application level session models.

2.8.2.2 Communicating Finite State Machines. While synthesis of Communicating Finite State Machine (CFSM) from execution traces has been studied we did not find any attempts at CFSM inference from traces. Saleh [231] proposes a semi-automatic approach to reverse engineering a communications protocol that can synthesize a CFSM model of a protocols automata from execution traces. A network of CFSM consists of a set of FSM which communicate asynchronously over FIFO channels by sending and receiving typed messages [39, 196]. Each protocol entity is represented by a CFSM with error-free simplex channels represented by unbounded FIFO queue [39]. CFSM representation is useful for our problem domain because they can be checked for non-progress properties by reachability analysis and reverse reachability analysis [196]. Although CFSM synthesis algorithms are presented by [231] we did not discover any systems that implement CFSM synthesis or inference.

2.8.2.3 n -Gram and Word Models. The n -gram and word models techniques presented by Rieck and Konrad are focused on anomaly detection for intrusion detection purposes [216]. In [216] an incoming connection payload x corresponds to consecutive sequence of symbols from an alphabet Σ . The content of x can be modeled as a set of subsequences w taken from the language $L \subseteq \Sigma^*$. The length of w is denoted by n . The model on n -grams can be derived by defining $L = \Sigma^n$. L is the language containing all sequences of fixed length n . Provided a set of delimiter symbols $D \subset \Sigma$, the model of *words* defined as $L = (\Sigma \setminus D)^*$ where every $w \in L$

subsequence of x is delimited by symbols from D . The chosen language L constitutes the basis for calculating similarity between network connections.

This could be a useful means of converting an input stream into a vector of values which can be used as a basis of comparison in machine learning techniques such as kernel methods used by Clark [45, 46] (See Section 3.3.2.2).

The n -gram approach strongly parallels grammatical inference techniques. In fact, stochastic k -TS models are equivalent to n -grams, with $n = k$ [271, Fact 1]. Furthermore, n -gram models have been combined with GI techniques such as MGGI [271] using k -TS representation and restricted k -TSS automata [268] (See Section 3.10.4.1).

2.8.2.4 Other Approaches. Communicating-X machines [16, 17, 128] are another possible formal representation but they have not received wide treatment in respect to formal performance analysis. We did not discover any attempts at recovery of models as Communicating-X machines. Another formal representation, Event-Driven Extended Finite State Machine (EEFSM), is presented in [142]. Lee formally associates the data portion in EEFSM as variables and parameters [142]. The EEFSM construct is used to develop passive testing algorithms for the OSPF and TCP state machines [142].

One practical application is ScriptGen which is an automated script generation tool for the Honeyd virtual honeypot ¹⁰ [147]. It is designed to monitor, capture, and analyze packets used by unknown protocols then generate scripts for replay in a honeyd honey pot environment. Unfortunately, the authors do not reveal the particulars of their implementation.

2.9 Chapter Summary

In this chapter we provided an English languages description of the problem domain under consideration. Next we discussed distributed systems and issues related

¹⁰Honeyd Virtual Honeypot – <http://www.honeyd.org/>

to protocol design recovery. Finally, we introduced protocol reverse engineering and overviewed state-of-practice and state-of-art reverse engineering tools and techniques.

III. Algorithm Domain

This chapter relates the problem domain of dynamic protocol reverse engineering from network traces to the algorithm domain of grammatical inference. We introduce Chomsky Hierarchy as a framework for discussing computational learnability. Next, we develop the symbolic model and mathematical notation that succinctly defines the characteristics of the algorithm domain. Finally, we discuss several existing algorithmic and heuristic approaches to grammatical inference.

3.1 Design Recovery from Samples of Behavior

Design recovery from samples of behavior has been studied for several purposes: automated specification mining of instrumented software executables [6] and Java object behavior mining [59]. Process discovery from samples of behavior (in event logs) is also studied for discovery of software process models [53] and workflow discovery [202, 235].

3.2 A Language Recognition Problem?

We refer back to our investigative questions presented in Section 1.5:

- IQ1** What information is necessary to reverse engineer the control portion of application layer protocols from data flows?
- IQ2** Given the proven [7, 95] difficulty of inferring finite automata from positive samples only, are there GI approaches that are appropriate for reverse engineering automata representations of the control portion of application layer protocols from data flows?

Bhargavan [24] formulates the problem of monitoring interactive devices like network protocols as a language recognition problem. The authors propose that given a specification that accepts a certain language of input-output sequences we can define another language that corresponds to the externally observable input-output sequences [24]. In essence, can we recover the model of a protocol given examples

of its behavior? Or more specifically, can we algorithmically turn application level network traces into automata?

With this background in mind we present aspects of formal language theory that frame the discussion of the algorithm domain under consideration.

3.3 Formal Languages

While protocol forward engineering efforts can utilize a range of formal models to represent the protocol under design they often do not [231]. We present aspects of formal languages to frame the discussion of model recovery for reverse engineering purposes.

3.3.1 Chomsky Hierarchy. The Chomsky Hierarchy, devised by Noam Chomsky, as presented in Table 3.1 and Figure 3.1 is a widely accepted framework for the discussion of formal representation of grammars and their expressive power. Informally, a sentence is a string of symbols, a language is a set of sentences, and a grammar is a (finite) list of rules defining the language [184].

Three basic decision problems for the representations are the questions of [3,110]:

membership - is a given sample sentence a member of a language?

equivalence - are two grammars able to recognize the same language?

emptiness - is the representation empty?

Angluin further expands this list to include the decision problems of [11, Section 1]:

subset - are all elements of a given language also members of another language?

superset - does a given language contain all elements of another language?

disjointness - do two languages have no element in common?

exhaustiveness - is there a member of a given sample that is disjoint from the possible languages?

The membership problem for Type-0 is undecidable, Type-1 is decidable, Type-2 can be determined in polynomial time, and Type-3 in linear time. The equivalence of Type-0, Type-1, and Type-2 are undecidable. Type-3 can be calculated in polynomial time only when the representation is a DFA¹. Even with these restrictions grammars are useful for studying languages. Grammars give a compact representation that supports recursivity. Also, grammars support graphical representations such as automata and parse trees (see Figure 3.2). Finally, even the “easiest” class, Type-3, contains **SAT**, boolean functions and parity functions [254].

The representational power of the Chomsky Hierarchy is Type-3 \subset Type-2 \subset Type-1 \subset Type-0. While Type-0, Type-1, and Type-2 based models have higher representational power they are more challenging to evaluate for performance characteristics. There are no efficient means known for generating parsers for Type-0 or Type-1 languages. Type-2 grammars can be parsed using Earley’s algorithm with $O(n^3)$ time for ambiguous grammars, $O(n^2)$ for unambiguous, and $O(n)$ for subcategories [71]. Subcategories of Type-2 grammars that are extensively studied include k symbol look-ahead parsers like the bottom-up $LR(k)$; and top-down, recursive descent and $LL(k)$ parsers. Where the expressive power is: $LL(k) \subset LR(k) \subset$ Type-2 [224, Vol 1, Chap 3, Sec 6.8].

Real data, such as network traces, often directly correspond to more complex languages. Application level protocols, like HTTP [80] and SMTP [113], are frequently specified in Backus-Naur Form (BNF)² or Augmented BNF [57].

Even so, it is possible to approximate a protocols language with simpler formal languages. Context-free languages can be approximated by algorithmically generated DFA [174, 181, 182]. DFA have low representational power but can be analyzed for performance characteristics within combinatorial limits.

¹ For example, the equivalence problem for a Type-3 finite-state transducers is undecidable.

² *Backus normal form* or *Backus-Naur form* [130] is a widely used alternative representation for context-free grammars.

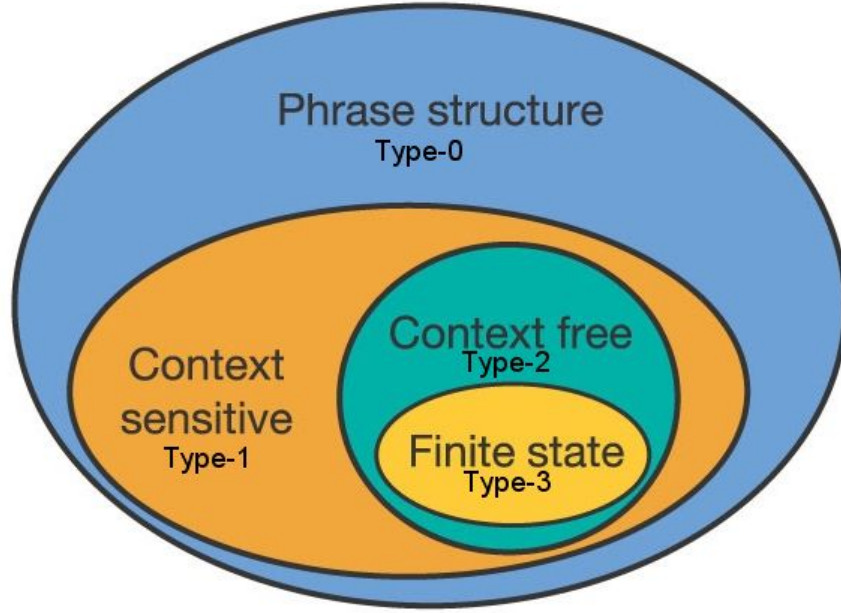


Figure 3.1: Chomsky Hierarchy [184].

Table 3.1: Chomsky Hierarchy. [159]

Chomsky Hierarchy			
Type	Languages	Automaton	Production rules
Type-0	Recursively enumerable	Turing Machine (unrestricted or phrase structure)	$\alpha \rightarrow \beta$ ($\alpha, \beta \in (V \cup \Sigma)^*$, α contains a variable)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing Machine	$\alpha \rightarrow \beta$ ($\alpha, \beta \in (V \cup \Sigma)^*$, $ \beta \geq \alpha $, α contains a variable)
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \alpha$ ($A \in V, \alpha \in (V \cup \Sigma)^*$)
Type-3	Regular	Finite state automaton	$A \rightarrow \alpha B, A \rightarrow a$ ($A, B \in V, a \in \Sigma$)

Table 3.2: SMTP Sender Transitions.

Operators (Σ)	States (Q)			
	INITIAL (q_0)	CONN_ESTABLISHED	TRANSACTION_STARTED	DATA_TRANSFER
DATA	-	-	DATA_TRANSFER	-
HELO	-	CONN_ESTABLISHED	TRANSACTION_STARTED	-
MAIL	-	TRANSACTION_STARTED	-	-
NOOP	-	CONN_ESTABLISHED	TRANSACTION_STARTED	-
QUIT	-	INITIAL	INITIAL	-
RCPT	-	-	TRANSACTION_STARTED	-
RSET	-	-	CONN_ESTABLISHED	-
end-of-data*	-	-	-	CONN_ESTABLISHED
more-data*	-	-	-	DATA_TRANSFER
open*	CONN_ESTABLISHED	-	-	-

Protocol state can be represented by grammars, such as context-free or finite state automata, drawn from the Chomsky Hierarchy. If we select Type-3, in effect, each endpoint of the communication is a tuple $A = \langle Q, \Sigma, \delta, q_0, F \rangle$. Q is a finite set of states. Σ is a finite set of symbols to label the transitions, known as the alphabet. Σ is the finite set of verbs or *operators* in the protocol vocabulary that lead to state transitions. Protocol operators must follow the syntactical format of the protocol. The results of protocol operators are the transitions represented by δ . Where δ , the partial mapping from $Q \times \Sigma$ into Q , represents all transitions.

For a simple finite automaton representation of a protocol we define the two endpoints of a protocol as the sender represented by tuple $S = \langle Q_s, \Sigma, \delta_s, q_{s0}, F_s \rangle$ and receiver represented by $R = \langle Q_r, \Sigma, \delta_r, q_{r0}, F_r \rangle$. Note that S and R share the same set of operators Σ . The structure of a distributed systems communication is determined by the syntactic format of the protocol operators that make up the finite set Σ .

As an example Figure 3.2 shows the states and operators for the sender of a Simple Mail Transfer Protocol (SMTP). The figure depicts four states, the operators and resulting state transitions.

Mapping Figure 3.2 to a DFA results in the following:

$$\begin{aligned}
 Q &= \{INITIAL, DATA_TRANSFER, CONN_ESTQABLISHED, TRANSACTION_STARTED\} \\
 \Sigma &= \{HELO, NOOP, QUIT, MAIL, RSET, END_OF_DATA, RCPT\} \\
 q_0 &= \{INITIAL\} \\
 F &= \{INITIAL\}
 \end{aligned}$$

The transition function represented by δ is shown in Table 3.2.

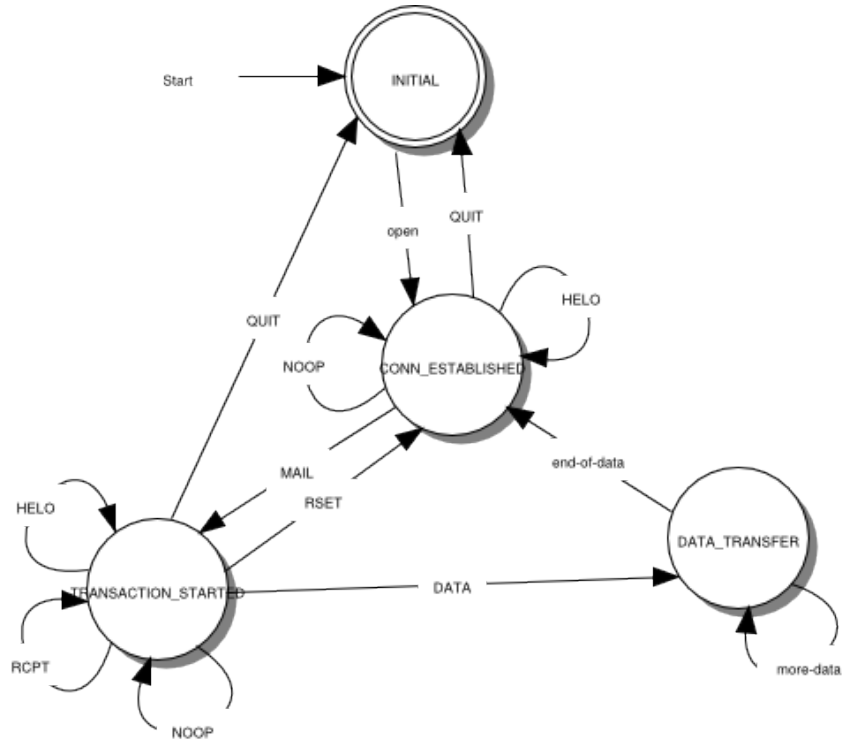


Figure 3.2: State Diagram for SMTP Sender.

It must be noted that the operators marked with * in Table 3.2 indicate transitions that must be inferred from the TCP transport service level even though they are shown in Figure 3.2. Also, the DFA representation does not have to account for issues such as connection loss or timeout relegating these issues to the underlying TCP transport layer.

3.3.2 Other Formal Representations. Alternative frameworks for language representation include pattern languages [8] [224, Chapter 6] and categorical grammars [124]. Various other constructs have been proposed that parallel and cross-cut the Chomsky hierarchy. Two examples are Petri nets and planar languages.

3.3.2.1 Petri nets. Petri nets are another mechanism for representing communicating systems that have analogy to the Chomsky hierarchy. Petri nets have high representational power but formal performance analysis can be more difficult. Petri net variations were used by van der Aalst [266] for workflow mining and dis-

covering social networks [265]. Mayo [162] proposes a hill-climbing technique to learn Petri net models of gene interactions.

3.3.2.2 Planar languages. Clark presents grammatical inference of planar languages using string kernel methods in [45, 46]. Planar languages cut across the Chomsky Hierarchy for simple-subsequence kernels [46]. The method was able to learn some context-sensitive and mildly context-sensitive languages. Artificial data sets and **Matlab**[®] source code are available at the Grammatical Inference with String Kernels project web site [43].

3.4 Learning Automata Representation of a Language

The process of learning an automata can be expressed as a decision problem: Given an integer n and two disjoint sets of words S_+ and S_- over a finite alphabet Σ , does there exist a DFA consistent with S_+ and S_- with a number of states less than or equal to n . The learning process is defined formally in Definition 3.4.1.

Definition 3.4.1 (Grammar Induction [41]). A general definition of *grammar induction* is given sets of labeled example string S_+ and S_- such that $S_+ \subset L(G)$ and $S_- \subset L'(G)$ infer a DFA (A) such that the language of A denoted $L(A) = L(G)$ is a language generated from an unknown Type-3 grammar (G). Its complement, $L'(G)$, is defined as $L'(G) = \Sigma^* - L(G)$ where Σ^* is the set of all strings over the alphabet (Σ) of $L(G)$.

Learning automata representation of languages by grammar induction is a widely researched topic. Miclet provides an introduction in [36, Chapter 9]. A survey to 1994 is presented by Vidal in [270]. A contemporary (2005) bibliographic survey of the field is presented by de la Higuera in [61].

3.5 Computational Learnability Models

There are three major formal models established in the computational learning community for learning grammar structure from examples or *grammatical inference*:

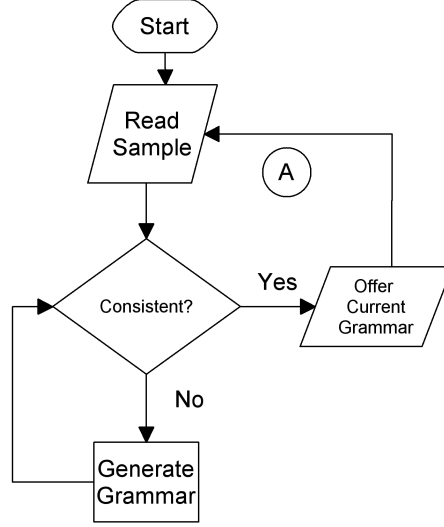


Figure 3.3: Gold’s Enumeration Procedure - the decision labeled *Consistent* determines if the current grammar is consistent with the sample presented so far. Once the learner enters loop A it converges to the target language in the limit [201, Figure 1].

Gold’s identification in the limit, Angluin’s query learning model, and Valiant’s probably approximately correct (PAC) learning model.

3.5.1 Identification in the Limit. Gold [95] proposed his identification in the limit model in the late 1960’s. Gold’s model describes a learning process where an infinite sequence of examples from a grammar G is presented to the inference algorithm M . Figure 3.3 outlines Gold’s enumeration procedure. The decision labeled *Consistent* determines if the current grammar is consistent with the sample presented so far. Once the learner enters the loop, denoted A, it converges to the target language in the limit. The eventual limiting behavior of the algorithm is used as the criterion of its success. Gold also shows that there is no general method of language inference from positive samples that will do better than enumeration [95]. Although Gold establishes the theoretical tractability of grammar induction from positive samples he does not provide algorithmic methods other than exhaustive enumeration.

3.5.2 Query Learning Model. Angluin developed the L^* algorithm to learn regular languages based on queries and counterexamples [10]. The inference algorithm is assumed to have access to an expert teacher, like an oracle. The teacher can answer specific queries, membership and equivalence, asked about an unknown grammar G . The teacher answers a *membership query* with an input string $w \in \Sigma^*$ with an output of “yes” if w is generated by G and “no” otherwise. An *equivalence query* takes an input grammar G' and the output is “yes” if the G' generates the same language as G and “no” otherwise. In the case where the answer is “no” a string w in the symmetric difference of the language $L(G)$ generated by G and the language $L(G')$ generated by G' is returned. The returned w is a counterexample.

In the inference from a protocol trace the equivalence test can at best be approximated while membership queries can be answered by testing the protocol under inspection [101, 247].

Angluin’s query learning method is extensively studied. Tradeoff of equivalence and membership queries is discussed by Balcázar et al [14]. A proof technique for demonstrating the hardness of learning by queries regardless of representation is established by Aizenstein, Hegedűs, Hellerstein and Pitt in [4]. Raffelt [211] further extends the L^* algorithm to deal with Mealy machines. The problem of identifying a value $\epsilon > 0$, where $1 - \epsilon$ is the probability that the oracle answers correctly (or if already asked, consistently) is left as an open question in the field of grammatical inference [62].

3.5.3 PAC Learning Model. Valiant [263] introduced probably approximately correct learning which is a distribution independent probabilistic model of learning from random examples. The inference algorithm takes a sample as input and produces a grammar as output. A successful inference algorithm is one that with high probability (at least $1 - \delta$) finds a grammar whose error is small (less than ϵ). Haussler provides an introduction to PAC learning in [102]. Furthermore, Angluin has shown that an equivalence query algorithm can be translated into a PAC learning

model [11, Section 2.4]. In fact, DFA are not PAC learnable unless we are allowed to ask membership queries on an oracle [11].

3.6 *Tractability*

Gold [95] and Angluin [7] proved that when using passive learning the problem of finding the smallest automaton consistent with a set of accepted and rejected strings is **NPC**. Golds Theorem states that inference on all regular languages is impossible with only positive samples. Gold further showed that exact identification from sparsely labeled samples is **NPC**. The difficulty of the problem was further established by Pitt and Warmuth [203,205] as well as Pitt and Valiant [204]. While this does not prevent a solution it does mean the solution will likely require approximation or heuristic techniques. In fact, Lang [137] demonstrated experimental evidence that the average case is tractable and Freund et al [88] proved the average case is polynomial.

3.7 *Search Approaches for Grammar Induction*

Given the tractability of grammar induction from positive samples regardless of representation Vidal proposes three classes of search for grammar induction: methods that use additional information, characterizeable methods, and heuristic methods [271]. We re-define Vidal's first class as extrinsic methods. For *extrinsic methods* the additional information extrinsic to the target model includes negative samples, equivalence queries or probabilistic information. While *characterizeable methods* concentrate on subclasses of regular languages that are shown to be learnable from positive samples. And, finally, *heuristic methods* which make direct use of *a priori* intrinsic knowledge of the target model such as the syntactic constraints of the language.

Muggleton also discusses the use of additional information (i.e. negative samples, limiting the number of states in target automata, or assigning statistical values to rank target automata [177, p.121].) Muggleton proposes the use of semantic information, similar to what we term intrinsic information, from the positive samples [177, Section 6.7].

The selection of search approach methodology is driven by what is known about the language under consideration. A composition of the three approaches might be appropriate if we have more than one type of knowledge (extrinsic, characterizeable, or intrinsic) available.

3.8 Notations and Definitions

Before we discuss specific algorithms we present the formal notation and definitions that will be used³. The definitions provided are derived from [9], [177, Appendix B], [271], [131], [37], [55], [41], and [62]. In general we favor the format used by [9] and [55]. Formally defining the mathematical symbology allows us to discuss the selected algorithm domain in a more compact form.

Definition 3.8.1 (Strings [62]). A *string* w over Σ is a finite sequence $w = a_1a_2a_3 \dots a_n$ of letters. Let $|w|$ denote the length of w . Letters of Σ will be indicated by a, b, c, \dots , strings over Σ by u, v, \dots, z , and the empty string by λ . Let Σ^* be the set of all finite strings over alphabet Σ

Definition 3.8.2 (Languages [62]). A *language* L is any set of strings, so therefore $L \subseteq \Sigma^*$. Operations over languages include: set operations (union, intersection, complement); product $L_1 \cdot L_2 = \{uv : u \in L_1, v \in L_2\}$; powerset $L^0 = \lambda L^{n_1} = L^n \cdot L$; and star $L^* = \cup_{i \in \mathbb{N}} L^i$. We denote by \mathcal{L} or \mathcal{A} a class of languages.

Algebraic laws for languages are discussed in texts on formal languages (e.g. [3, 110, 159, 224]).

Definition 3.8.3 (Learning Sample of a Language [55]). A *learning sample* S of a language L is a finite multi-set of words from L . That is $\forall w \in S, w \in L$.

Definition 3.8.4 (Finite State Automaton). A *finite state automaton* (FSA), A is a quintuple $A = \langle Q, \Sigma, \delta, I, F \rangle$, where Q is a finite set of states, Σ is a finite set of

³The reader is referred to texts on formal languages (e.g. [3, 110, 159, 224]) if they require background detail.

symbols to label the transitions, known as the alphabet, δ is a partial mapping from $S \times \Sigma \rightarrow S$, $I \subset Q$ is the set of start states, and $F \subset Q$ is the set of final states.

The size of the automaton is defined as the total number of states in Q denoted by $|Q|$, that is $|A| = |Q|$.

$Pref_A(q)$ will denote the prefix language of a state q defined by $Pref_A(q) = \{w \in \Sigma^* | q \in (q_0, w)\}$ [55, Definition 1].

$Suff_A(q)$ will denote the suffix language of a state q defined by $Pref_A(q) = \{w \in \Sigma^* | \delta(q, w) \cap F \neq \emptyset\}$ [55, Definition 1].

The FSA is *minimized* if no pair of states are equivalent. Given $q_i, q_j \in Q, i \neq j$, there is an input word x that distinguishes them such that $\delta(q_i, x) \neq \delta(q_j, x)$.

Definition 3.8.5 (Deterministic Finite State Automaton). A FSA is deterministic or a *deterministic finite state automaton* (DFA) $\forall q \in Q, \forall a \in \Sigma, \delta(q, a)$ has at most one element otherwise the FSA is non-deterministic. Additionally, $|I| = 1$ with the start state denoted by q_0 where q_0 is the single element of I .

Definition 3.8.6 (Acceptance [55]). An *acceptance* for a word $w \in \Sigma^*$, where $w = a_1 a_2 a_3 \dots a_{|w|}$, in an automaton $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ is a sequence $(q_0, \dots, q_{|w|})_w$ of $|w| + 1$ states such that $q_0 \in Q, \forall i \in [1, |w|], q_i \in \delta(q_{i-1}, a_i), q_{|w|} \in F$. q_0 is said to be the initial state and $q_{|w|}$ is said to be the final state of the acceptance. Transitions q_{i-1}, a_i, q_i are said reached by the acceptance. The set of acceptances of a word w in automaton A is denoted by $Acc_A(w)$.

Definition 3.8.7 (Regular Grammar). A *regular grammar* is a grammar in which all of the productions are of the form $A \rightarrow aB$ or $A \rightarrow \lambda$. Given a finite automaton $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, it is possible to construct a regular grammar $G = \langle N, T, P, S \rangle$ such that $L(A) = L(G)$ as follows. For each state in Q add a non-terminal of the same name to N . For each symbol in Σ add an equivalent symbol to T . Let S be the non-terminal named q_0 . For each $q \in F$ add a production to P of the form $q \rightarrow \lambda$. For each transition of the form $\delta(q_1, a) = q_2$ add a production to P of the form $q_1 \rightarrow aq_2$.

Definition 3.8.8 (Regular Languages). The transition function δ of a DFA can be extended to Σ^* : $\delta(q, \lambda) = q$ and $\delta(q, a.w) = \delta(\delta(q, a), w)$ for all $q \in Q$, $a \in \Sigma$, $w \in \Sigma^*$. Let $\mathbb{L}(A)$ denote the language recognized by the automaton A : $\mathbb{L}(A) = \{w \in \Sigma^* | \delta(q_0, w) \in F\}$ By definition the language $\mathbb{L}(A)$ accepted by a DFA A is a regular language. That is the class of regular languages can be defined as the class of languages accepted by a finite automata.

Definition 3.8.9 (Type-3 Grammar [41]). A *Type-3 Grammar* is a four-tuple $G = \langle T, N, P, S \rangle$ where:

- $T \subseteq N$ is a finite non-empty set called the terminal alphabet of G ,
- N is a finite non-empty set called the total vocabulary of G ,
- P is a finite set of production rules,
- $S \in (N - T)$ is referred to as the start state, and the rules in P are of the form $A \rightarrow aB$ or $A \rightarrow a$, where $A, B \in (N - T)$ and $a \in T$

Definition 3.8.10 (Canonical Automaton [55]). A *Canonical Automaton* (CA) of a language L , denoted by $CA(L)$ is the sole minimal DFA accepting L .

Definition 3.8.11 (Universal Automaton [55]). A *Universal Automaton* (UA) of a language L , denoted by $UA(L)$ is the canonical automaton $A(\Sigma^*)$ accepting all words $w \in \Sigma$.

Definition 3.8.12 (Maximal Canonical Automaton [55]). A *Maximal Canonical Automaton* (MCA) related to a learning sample S , denoted by $MCA(S)$ or more simply MCA , is the union for each word w of learning sample S from a canonical automata $A(\{w\})$. A MCA is a star-like automaton that exactly recognizes data from sample S .

Definition 3.8.13 (Partition of set S [177]). A *partition of set S* , denoted by π_S , is a set of pairwise disjoint non-empty subsets of set S s.t. the union of all π_S is equal to S .

Definition 3.8.14 (Block [177]). The unique *block* of π_S containing s , where $s \in S$ is denoted $B(s, \pi_S)$.

Definition 3.8.15 (Refines [177]). Given two partitions, π and π' π *refines* π' iff every block of π' is a union of blocks of π .

Definition 3.8.16 (Prefix Tree Acceptor [55]). A *Prefix Tree Acceptor* (PTA) on a sample S , $PTA(S)$ or PTA , is the deterministic automaton obtained when merging every state of $MCA(S)$ with identical prefix languages.

A PTA is a FSA that can be constructed by laying out the strings in a language using a state to represent each unique prefix of one of the strings. A language accepted by a PTA is regular and exactly accepts all strings in a given language.

Figure 3.4 shows the PTA for POP3 client commands sent to servers from week 1 day 1 inside IDEVAL traffic.

Definition 3.8.17 (Augmented Prefix Tree Acceptor [41]). A *Augmented Prefix Tree Acceptor* (APTA) is a six-tuple $G = \langle Q, \Sigma, \delta, s, F_+, F_- \rangle$ where:

- Q is a finite non-empty set of nodes,
- Σ is a finite non-empty set of input symbols or input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ the transition function,
- $s \in Q$ the start or root node,
- $F_+ \subseteq Q$ identifies final nodes of strings in S_+ ,
- $F_- \subseteq Q$ identifies final nodes of strings in S_- ,

The size of an APTA is defined as the total number of elements in Q denoted $|Q|$.

Definition 3.8.18 (APTA node equivalence [41]). Two nodes q_i and q_j in an APTA are considered not equivalent if and only if:

- $(q_i \in F_+ \wedge q_j \in F_-) \vee (q_i \in F_- \wedge q_j \in F_+)$, or

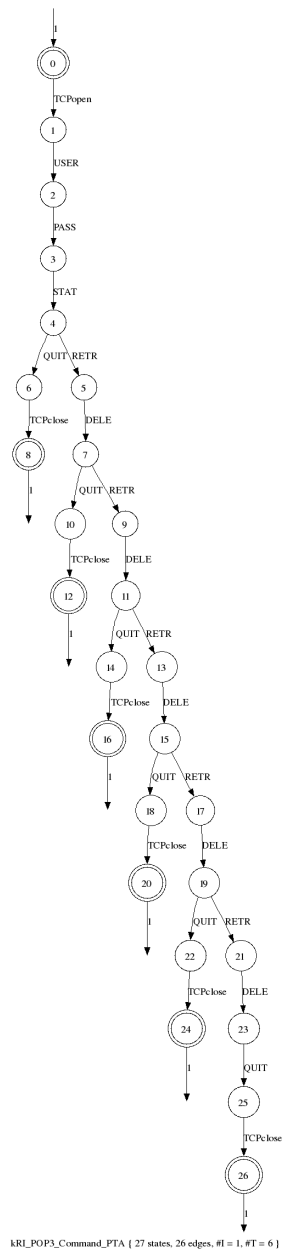


Figure 3.4: Example PTA - Command PTA for POP3 from Week 1 Day 1 inside IDEVAL traffic.

- $\exists s \in \Sigma$ such that if $(q_i, s, q_{i'}) \in \delta, (q_j, s, q_{j'}) \in \delta$ then $q_{i'}$ is not equivalent to $q_{j'}$.

Definition 3.8.19 (Strictly locally testable languages [37]). Let k be a positive integer. For $w \in \Sigma^+$ of length $\geq k$, let $L_k(w), R_k(w)$, and $I_k(w)$ be respectively the prefix length k , the suffix length k and the set of interior factors of length k of the word w . $L \subseteq \Sigma^*$ is strictly k -testable if and only if there exist three sets X, Y, Z of words on Σ such that for all $w \in \Sigma^+, |w| \geq k$, $w \in L$ iff $L_k(w) \in X, R_k(w) \in Y$, and $I_k(w) \subseteq Z$. A language is *strictly locally testable* if it is strictly k -testable for some $k > 0$. We denote the class of strictly locally testable languages as $s\mathcal{LT}$.

Definition 3.8.20 (Left Quotient [9, 131, 177]). Let L be any language. $Pre_{(L)}$ The set of all prefixes of elements of language L .

For any $w \in \Sigma^*$, we denote the *left-quotient* of language L and word w by $w \setminus L$. That is $w \setminus L = \{x \in \Sigma^* | wx \in L\}$. Angluin introduced the equivalence relation \cong_L over Σ^* defined as: $w_1 \cong_L w_2$ iff $w_1 \setminus L = w_2 \setminus L$. A language L is regular iff the number of equivalence classes of \cong_L is finite.

Definition 3.8.21 (k -tails [177]). The k -tails of word w in language L denoted by $w \setminus^k L$ is the set $\{v : v \in w \setminus L, |v| \leq k\}$. That is, the k -tail is the set of all words in the language that are members of the left quotient with a depth of no more than k .

Definition 3.8.22 (k -reversible Languages [9, 131]). A language L is *pseudo k -reversible* iff whenever u_1uw and u_2uw are in L and $|v| = k$, $u_1 \cong_L u_2v$ holds. A language L is k -reversible iff L is regular and pseudo- k -reversible. For any non-negative integer k , we denote the class of k -reversible languages by \mathcal{Rev}_k . It is known that for any non-negative integer k , the class \mathcal{Rev}_k is properly contained in the class \mathcal{Rev}_{k+1} .

3.9 Grammatical Inference Algorithms

Several algorithms exist for inferring grammars for language understanding. The algorithms have been used in a range of GI tasks including machine learning, for-

Table 3.3: Regular Inference Algorithms.

Regular Inference Algorithms		
Algorithm	Negative Samples Required	Target Automaton
Trakhtenbrot and Barzdin	Yes	DFA
ECGI	No	DFA, PFSA [269, Section 2.2]
k -TSSI	No	k -testable DFA
k -RI Angluin	Optional	k -reversible DFA
k -RI Muggleton	Optional	uniquely τ -terminated k -reversible DFA
MGGI	No	DFA
RIG and BRIG	Yes	DFA
RPNI	Yes	DFA
EDSM	Yes	DFA

mal language theory, structural recognition, natural language processing, and speech recognition [61].

Algorithms can be classified according to their target language/grammar: Type-3, regular; Type-2, context-free; Type-1, context-sensitive; or Type-0, phrase structure (see Figure 3.1). Algorithms can also be characterized according to the classes discussed in Section 3.7, that is: extrinsic, characterizable, or heuristic. The main types of extrinsic information we are concerned with are negative samples, and queries.

3.10 Inference of Regular Languages (Type-3)

Inference of regular languages is well studied. Muggleton provides an introduction to regular inference in [177, Chap 6]. Gronfors [99] conducts experimental analysis of several of algorithms for generality. Dupont presents an early look at the search space of regular inference in [70]. Hingston describes various approaches to develop a family of regular inference algorithms [106]. Coste and Fredouille provide a discussion of the search space of inference of DFA, NFA, and unambiguous finite automaton in [55]. While Coste et al [54] discuss the importance of domain and typing background knowledge to tune inference algorithms.

The limitations on inference of the union of multiple languages from intermixed samples is discussed by [278] who refines Angluin’s necessary and sufficient conditions for inference. Given that we do not know if application protocol languages meet the conditions defined by [278] we will not further consider intermixed samples for bilingual inference.

Table 3.4: Regular Inference Algorithm Performance.

Regular Inference Algorithm Performance		
Algorithm	$O()$	Notes
Trakhtenbrot and Barzdin	$O(mn^2)$	where $m = \text{initial APTA} $, $n = \text{number of states in the final hypothesis automaton.}$
ECGI	not characterized	experimental evidence from [227].
k -TSSI	$O(kn \log n)$	where n is the sum of the lengths of all the strings in S_+ [89].
k -RI Angluin	$O((k+1)^2 n^3)$ [9, Theorem 35]	where n is the sum of the lengths of all strings in the sample.
k -RI Muggleton	$O(n^2)$ [177, Section 6.5]	where n is the sum of the lengths of all strings in the sample.
MGCI	not characterized	
RIG and BRIG	non-polynomial	experimental evidence from [169, Section 3.5]
RPNI	$O((m+m')m^2)$	where m is the sum of the length of all strings in S_+ and m' is the sum of the length of all strings in S_-
EDSM	not characterized	

Table 3.3 summarizes the characteristics of several algorithms used for inference of regular languages. Table 3.4 summarizes the performance characteristics of several of the algorithms discussed.

3.10.1 Statistical Extrinsic Methods. Muggleton provides a framework to represent several statistical extrinsic methods to determine state merges by defining a generalized algorithm using a predicate function $\chi(u, v)$ [177, Appendix C]. The generalized algorithm is shown in Algorithm 1, where $\chi(u, v)$ takes values shown in Table 3.5 for some k -tails algorithms.

Input: S_+ non-empty set of positive sample strings	
Output: The acceptor $A_0/\pi_{Pr(S_+)}$	
/*Initialization	*/
/* A_0 is a DFA represented by $\langle Q_0, \Sigma, \delta_0, I_0, F_0 \rangle$	*/
1.1 Let $A_0 = \text{FormPTA}(S_+)$;	
1.2 Let π_0 be the trivial partition of Q_0 ;	
1.3 Let $i = 0$;	
/*Merging	*/
1.4 for $\forall (u, v) \in Q_0$ do	
1.5	if $\chi(u, v)$ then
1.6	Let $B_1 = B(u, \pi_i)$;
1.7	Let $B_2 = B(v, \pi_i)$;
1.8	Let π_{i+1} be π_i with B_1 and B_2 merged;
1.9	Increment i by 1;
1.10	end
1.11 end	
/*Termination	*/
1.12 Let $f = i$;	
1.13 return The acceptor A_0/π_f	

Algorithm 1: Muggleton Algorithm IM1 [177, p.100]

Table 3.5: Muggleton Predicate Functions $\chi(u, v)$ for k -tails. [177, Appendix C]

Muggleton Predicate Functions for k -tails	
Source	$\chi(u, v)$
Biermann and Feldman [28]	$\chi(u, v) = \begin{cases} true & u \setminus^k S^+ = v \setminus^k S^+ \\ false & otherwise \end{cases}$
Levine [148]	$\chi(u, v) = \begin{cases} true & Stren(u, v) \geq Strn \\ false & otherwise \end{cases}$ <p>, where</p> $Stren(u, v) = \max_i \left[\frac{2 u \setminus^i S^+ \cap v \setminus^i S^+ }{ u \setminus^i S^+ + v \setminus^i S^+ } \right], i \in \mathbb{Z}^+$ <p>, and</p> $Stren[0, 1] \in \mathbb{R}$
Miclet [168]	$\chi(u, v) = \begin{cases} true & u \setminus S^+ \cap v \setminus S^+ \neq \emptyset \\ false & otherwise \end{cases}$

Another statistical extrinsic method, Minimal Descriptor Length (MDL) as presented by [146], also fits into the Muggleton predicate form but does not use k -tails. Instead

$$\chi(u, v) = \begin{cases} true & \{|DFA'| + |DFA'(S_+)|\} \leq \{|DFA| + |DFA(S_+)|\} \\ false & otherwise \end{cases}$$

, where DFA is initially the PTA of S_+ and DFA' is the hypothesis DFA. DFA' replaces DFA for each successful iteration.

3.10.2 Extrinsic Negative Sample Support Methods. Several algorithms exist that require extrinsic negative sample support including: Trakhtenbrot and Barzdin's algorithm; Miclet's Regular Inference of Grammars, and Regular Positive and Negative Inference.

3.10.2.1 Trakhtenbrot and Barzdin. One of the earliest algorithms was proposed by Trakhtenbrot and Barzdin⁴ [41]. We classify the algorithm as an

⁴We were unable to locate a copy of the original presentation by Trakhtenbrot and Barzdin in [256].

extrinsic method because it was designed for completely labeled data sets containing both positive and negative examples. According to Cicchello [41, Section 10.1.3]:

The upper bound on the runtime is mn^2 , where m is the total number of nodes in the initial APTA and n is the total number of states in the final hypothesis.

Cicchello presents a modification that supports use of the algorithm with incomplete training sets in [41].

3.10.2.2 Regular Inference of Grammars. Regular Inference of Grammars (RIG) requires extrinsic negative sample support [169]. Miclet also presents Boosted beam-search Regular Inference of Grammars (BRIG). BRIG also requires extrinsic negative sample support [169]. Miclet describes both RIG and BRIG as inefficient [169, Section 3.5] and concludes from experimental results that the algorithms are non-polynomial [169, Section 5].

3.10.2.3 Regular Positive and Negative Inference. Regular Positive and Negative Inference (RPNI) requires extrinsic negative sample support. The algorithm was proposed by Lang [137] and independently by Oncina and García [189]. The RPNI algorithm has been shown to identify in the limit regular languages. The complexity is a function of the positive and negative sample sizes. RPNI has an update time of $O((m + m')m^2)$ where m is the sum of the length of all positive data (S_+) and m' is the sum of the length of all negative data (S_-).

RPNI works by starting with the PTA, and merging pairs of states if possible, using a fixed depth-first ordering of state pairs. The algorithm runs in polynomial time and is guaranteed to identify the target FSA given complete sample data.

One limiting factor of the RPNI algorithm is that it requires presentation of the entire positive and negative sample data. If new data is available the inference process must be restarted. A modification to allow for incremental inference was proposed by Dupont [69]. The algorithm is modified to support noisy samples and presented as *RPNI** by [237]. More recently, Hoffman provides experimental evidence that

prohibiting some of the merges performed by the original RPNI algorithm improved performance against artificial random data sets [107].

3.10.2.4 Evidence Driven State Merging. Evidence Driven State Merging (EDSM) requires extrinsic negative sample support. The basic algorithm is described by Lang in [138]. EDSM performs merges in arbitrary order such that both nodes in a merge might be the roots of arbitrary subgraphs of the hypothesis automaton [138]. To overcome this a modification to EDSM called the blue-fringe algorithm restricts the merge order [138] using a policy described by [123]. The algorithm is further modified to support noisy samples and presented as *BLUE** by [237] as part of the Learning DFA from Noisy Samples competition [153] for the GECCO2004 conference [262].

3.10.3 Characterizeable Methods. Given Gold’s result that regular languages cannot be inferred in the limit from only positive data [95] the search for characterizable subclasses that can be inferred with only positive data has become a kind of “holy-grail” of grammatical inference. Many subclasses of regular languages have been proposed: strictly regular languages [284], k -reversible [9], locally testable languages in the strict sense [89], code regular languages [74], and Szilard languages of regular grammars [156, 282].

Figure 3.5 shows some of the families of languages that are classified within regular languages. Two well studied characterizable methods are k -Testable in the Strict Sense Inference and k -Reversible Inference.

3.10.3.1 k -Testable in the Strict Sense Inference. The inductive inference of the class of k -Testable languages in the strict sense was proposed by Garcia and Vidal in 1990 [89]. A language that is k -Testable in the Strict Sense of Inference (k -TSSI) is defined by a finite set of substrings of length k that are permitted in the target language [89]. This algorithm is a characterizable method that limits the target model to k -testable languages.

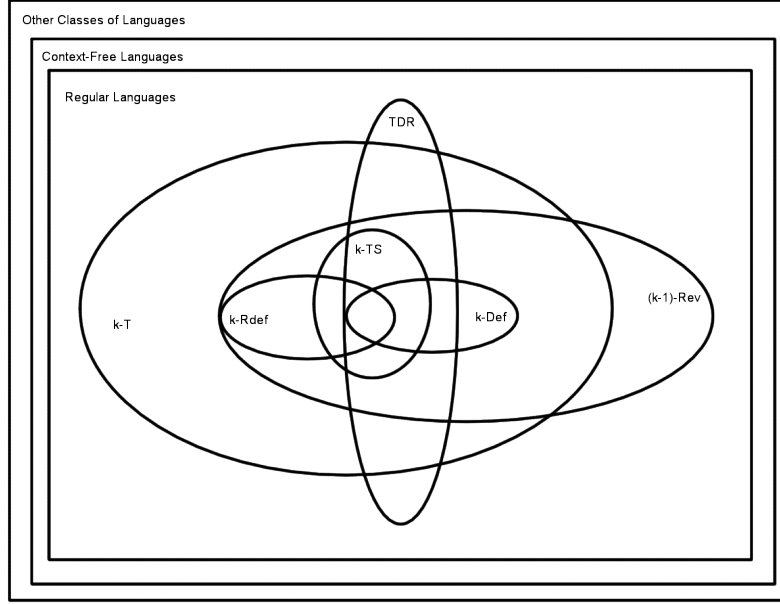


Figure 3.5: Some Families of Regular Languages [271].

A k -testable languages in the strict sense (k -TSSL) a subclass of k -testable languages. It is essentially defined by a finite set of substrings of length k that are permitted to appear in the strings of the language. Given a positive learning sample S_+ of strings of an unknown language, a deterministic finite-state automaton that recognizes the smallest k -TSSL containing S_+ is obtained.

The the number of transitions in the inferred automaton is bounded by $O(m)$ where m is the number of substrings defining the k -TSSL, and the inference algorithm works in $O(kn \log n)$ where n is the sum of the lengths of all the strings in S_+ [89, Theorem 6.1].

Torres and Varona [255] presents a low-level representation of k -TSS structures proposed for use in continuous speech recognition. Varona and Torres also conducted experimental analysis with k values of 4 and 5 on smoothed stochastic FSA for continuous speech recognition [268].

Experiments by [89, Section VIII] show the ability of (stochastic) k -TSSLs to approach other classes of regular languages. An algorithm for k -TSSI is outlined in [89, Figure 1].

3.10.3.2 k -Reversible Inference. k -Reversible Inference (k -RI), introduced by Angluin, does not require negative sample support [9]. Like, k -TSSI, k -RI is a characterizable method restricting the target automaton to k -reversible regular languages [9].

The class of k -reversible regular languages is a subset of the regular languages with the properties explained in Definition 3.8.22. The target language must be k -reversible for some $k \geq 0$. The k -RI algorithm identifies the minimum k -reversible language containing any finite positive sample in $O((k+1)^2 n^3)$ time, where n is the summation of the lengths of the strings in the sample [9, Theorem 35].

This was later reduced by Muggleton to $O(n^2)$ time with the added restriction that the target automaton is uniquely terminated, that is the automaton is a uniquely τ -terminated acceptor [177, Section 6.5]. A *uniquely τ -terminated acceptor* is a FSA with the property that any transition arc is labeled with the termination symbol τ if it leads to an acceptor state and that the acceptor state has no outgoing arcs [177, Section 6.5.1].

3.10.4 Heuristic Methods. Heuristic methods leverage intrinsic knowledge of the target automata. Heuristic methods concentrate on producing a target model that is useful for a problem domain not necessarily considering the automata’s membership in a language theoretic characterizable class. Algorithms in this category include: Morphic Generator Grammatical Inference, the Burge algorithm, Continuous Time Markov Chain models, and kBehavior.

3.10.4.1 Morphic Generator Grammatical Inference. Morphic Generator Grammatical Inference (MGGI) does not require extrinsic negative sample support. The inference procedure was introduced by Garcia as the “Local Language

Inference Algorithm” [90]. Sanchis discusses the use of MCGI for inferring phonetic units [232]. Vidal outlines the learning approach for MGGI regular inference in [271] but does not provide an algorithmic implementation.

MGGI works with two finite alphabets, Σ and Σ' and a set of positive sample strings $S_+ \subset \Sigma^*$ [271]. A function g is used to rename the words in S_+ resulting in $S'_+ = g(S_+)$ where $S'_+ \subset \Sigma'^*$ [271]. The corresponding $2-TS$ language is obtained from S'_+ and another renaming function h is applied [271]. The MGGI inferred language is $L = h(l(g(S'_+)))$. The renaming function is the morphic generator which allows for generalization of the language under consideration. The renaming function relies on extrinsic knowledge of the model under consideration.

We did not discover a formal analysis of the performance characteristics of MGGI. We did discover experimental empirical results specific to speech recognition in [232, 271]. Local language learning, similar to MGGI, is applied to DNA sequence analysis by [283].

3.10.4.2 Burge. The Burge algorithm⁵, which does not require negative samples, is presented by Ingham in [114, 115]. The algorithm is $O(nm)$ where n is the number of samples in the training set and m is the average number of tokens in a sample [114, Section 3.9]. Ingham presents modification to the algorithm to support incremental learning of DFA models from tokenized HTTP requests. While [115] does generate a notional model of the HTTP request the focus is on approximation of HTTP for intrusion detection not model recovery.

3.10.4.3 Continuous Time Markov Chains. Sen et al examine the use of grammatical inference inspired algorithms to learn edge labeled Continuous Time Markov Chains [238]. Java source code for their implementation is available.

⁵John Burge is a co-author of [115].

3.10.4.4 kBehavior. The k -tails approach is modified by Mariani and Pezzé and presented as kBehavior which is an incremental approach designed for limited storage capacity [158, Section 2]. The technique uses a heuristic to merge multiple states that are recognized as a common behavior instead of individual states. This is in part to improve branch and loop detection from execution traces.

3.10.5 Hybrid Methods. There are several randomized and heuristic approaches to regular language inference that do not neatly fit into our categories of extrinsic, characterizeable, or heuristic. If we have both intrinsic and extrinsic information hybrid methods are possible. One such method is Angluin’s L^* algorithm, another is Error Correcting Grammatical Inference.

3.10.5.1 Angluin’s L^ Algorithm.* Angluin also presents a modification to the k -RI algorithm using both extrinsic negative samples and queries [9, Section 7]. An overview of Angluin’s L^* algorithm is presented by Berg in [23]. Berg discusses evaluation of L^* (as presented by Angluin 1987 [10]) for prefix-closed DFA⁶ against random samples and real world examples drawn from the Edinburgh Concurrency Workbench⁷ [23].

3.10.5.2 Error Correcting Grammatical Inference. Error Correcting Grammatical Inference (ECGI) proposed by Rulot and Vidal is a GI heuristic that incrementally infers the target automata model [226]. ECGI combines statistical extrinsic methods and heuristic methods.

The approach, which does not require negative sample support, is based on error correcting parsing. The ECGI algorithm builds a hypothesis automaton by initially creating a trivial automaton from the first presented sample word [226]. States and transitions are added to the hypothesis automaton for every new unrecognized sample [226]. Error correcting parsing is used to determine what states and transitions to add

⁶A languages \mathcal{L} is *prefix-closed* if $\forall w \in \mathcal{L}$, then $\forall Pref_{\mathcal{L}}(w) \in \mathcal{L}$ [136, Definition 3.2].

⁷Edinburgh Concurrency Workbench – <http://homepages.inf.ed.ac.uk/perdita/cwb/>.

by searching for the best path for the input sample in the hypothesis automaton [226]. A statistical extrinsic method, such as Hamming or Levenshtein distance, can be used to measure which path is best. Heuristic restrictions eliminate loops and circuits in the inferred hypothesis automaton.

We did not discover a formal analysis of the performance characteristics of ECGI. We did discover experimental empirical results specific to speech recognition in [232].

The ECGI algorithm has also been applied in language modeling [209] and parts-of-speech tagging [206,207]. Sanchis also discusses the use of ECGI for inferring phonetic units [232]. Rulot also proposes an extension to the ECGI algorithm [226] to support stochastic target automata. The stochastic extension is expanded by [269, Section 2.2].

3.10.5.3 Other Approaches. Graine [97] introduces a method for learning regular languages with constant alphabet sizes using neural networks. The method is $O(n^2)$ time complexity for k -reversible regular languages. Giordano examines inference of regular languages by a tabu search that requires both positive and negative examples [92]. Belz proposes a genetic algorithm for automata inference [21]. Niparnan [183] and Lai [136] also examine genetic algorithm approaches to inference of finite automata.

3.11 Inference of Higher Order Languages

Grammatical inference of context-free languages (Type-2) has received some attention. Early work was conducted by [228] and [60]. Lee [145] provides a circa 1994 survey of literature to that point. The 2004 Omphalos Context-Free Language Learning Competition held in conjunction with the 7th International Colloquium on Grammatical Inference [246] generated experimental results for artificially generated data. More recently Oates [187] studied k -reversible CFG, Nakamura presented an

incremental CFG learning algorithm [179], while Petasis [197] and Javed [120] both propose genetic algorithm approaches.

Non-terminally separated languages (NTS) a subclass of deterministic context-free languages (where $\text{Type-3} \subset \text{NTS} \subset \text{Type-2}$) have also received some attention. Clark [44] recently examined PAC-learning of NTS languages. Another languages class that cross-cuts the Chomsky hierarchy is the class of very simple grammars proposed by Yokomori which contains elements of 0-reversible, left Szilard of linear, regular (Type-3), and NTS languages. [282].

We did not discover attempts to directly infer context-sensitive (Type-1) or recursively enumerable (Type-0) languages.

3.12 Chapter Summary

In this chapter we related the problem domain of dynamic protocol reverse engineering from network traces to the algorithm domain of grammatical inference. We introduced the Chomsky Hierarchy as a framework for discussing computational learnability. Next, we developed the symbolic model and mathematical notation that defines the characteristics of the algorithm domain. Finally, we discussed several existing algorithmic and heuristic approaches to grammatical inference.

IV. Experimental Design

The purpose of this chapter is to establish the methodology we will use to evaluate existing algorithms for effectiveness and efficiency in the dynamic protocol reverse engineering discipline to establish empirical evidence for the applicability of grammatical inference. A hybrid application of state-of-practice format recognition and grammatical inference is presented which could support the use of formal techniques to identify vulnerabilities in the specification, implementation, and deployed configuration of network protocols. We outline the approaches we implement in our experimental design and detail the results of format recovery and control flow recovery. The main emphasis is on techniques useful for protocol control flow (δ) recovery.

4.1 *Application Level Network Traces into Automata*

Once again, we must address the following four issues:

- Network trace collection.
- Application level protocol data flow recovery.
- Protocol format (Σ) recovery.
- Protocol transition function (δ) recovery.

None of the methods discussed in Chapter II or Chapter III provide an automated means of naming or uniquely identifying the operators that make up the vocabulary of an arbitrary protocol. For this reason we propose mining the protocols operator packet formats from existing open sources and algorithmically generating an automata representation. The choice of development tools and supporting toolkits are explained in Appendix B.

4.2 *Protocol Selection*

Specifications for open protocols, such as SMTP and POP3 are available in online specification documents. Using open protocols may seem counter-intuitive but it allows us to establish benchmarks for comparison. For closed or proprietary

protocols, open source projects such as Bro, jNetStream, and Wireshark embody the collective reverse engineering efforts of their contributors [19, 48, 140]. We select POP3 and SMTP because they are textually represented, synchronous, and the session boundaries are easily detected.

An additional reason we selected POP3 is that with only 10,960 TCP connection attempts on port 110 it is relatively low volume in relation to SMTP traffic with 126,545 TCP connection attempts. This allowed us to evaluate the proof of concept software on a lower volume, but similarly structured, protocol during incremental development.

4.3 *Algorithm Selection*

The problem domain we are considering has the following characteristics: we do not know if a sample is positive or negative, we do not have access to an oracle and we do not know if the languages under consideration are characterizable by regular languages or subclasses of regular languages. By constraining the scope of the problem to textually represented single-channel protocols using TCP transport on IPv4 networks (specifically SMTP and POP3) we know that the grammars for the protocol languages are specified English language and in a context-free format (Augmented BNF). We select k -RI, a characterizable method, and k -TSSI, an incremental characterizeable method for our proof of concept implementation.

4.4 *Experimental Architecture*

Our experimental architecture is simplified by the use of an existing data set and limiting the study to POP3 and SMTP. Because we are using an existing data set the network trace collection is already determined. Also, both POP3 and SMTP encapsulate a complete session within a single transport level TCP connection. This reduces our overall experimental architecture to protocol format (Σ) recovery and transition function (δ) recovery. Figure 4.1 shows the two components that we implemented.

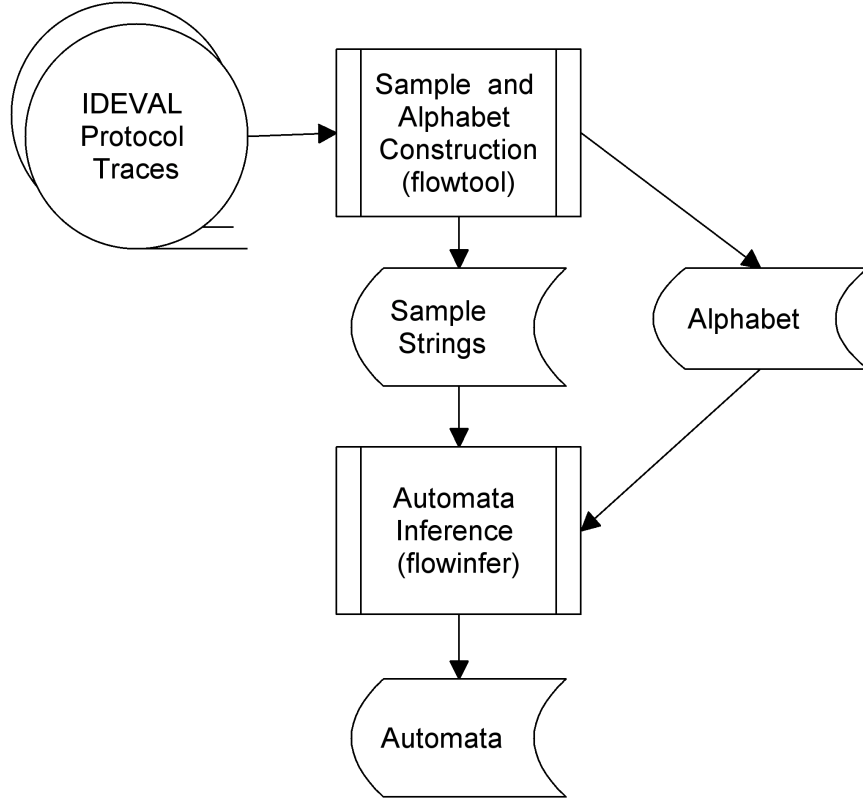


Figure 4.1: Experimental Architecture Overview - protocol format (Σ) recovery is implemented in the “Sample and Alphabet Construction” process. The low level implementation, called *flowtool*, uses hand coded POP3 and SMTP command and reply parsers to generate an alphabet and sample strings. Transition function (δ) recovery is implemented in the “Automata Inference” process. The low level implementation, called *flowinfer*, executes the inference algorithm on the alphabet and sample strings to generate an automata representation of the protocols control flow.

4.4.1 Network Trace Collection. Network trace collection for IPv4 protocols is well covered by others. For our experimental architecture we are using existing network traces so the collection architecture was pre-determined by the data set under investigation. The data set used is the IDEVAL data set discussed in Appendix A. The trace collection architecture used to build the IDEVAL data set is described in [151].

4.4.2 Application Level Protocol Data Flow Recovery. Both of the selected protocol’s session structures are encapsulated in single TCP connections. This makes extraction of application level session data flows equivalent to extracting TCP connections. Using the assumption that protocol ports are accurate for the IDEVAL data set we de-multiplex the raw data to extract TCP traffic on port 25 (SMTP) and port 110 (POP3). Finally, we merge the trace files for SMTP and POP3 traffic into a cumulative data file. The pre-processing workflow is detailed in Appendix A Section A.4.2.

4.4.3 Protocol Format Recovery. After the traces were extracted we used a tool we developed called *flowtool* to create alphabet and sample strings from the cumulative protocol traces and weekly protocol traces.

We implemented hand coded operator parsers for our proof of concept implementation. Since we are examining open protocols we were able to use the specification documents and heuristics from Wireshark and Bro to implement operator oriented parsers. We processed the network trace files data with our flowtool to extract the alphabet and sample strings. A low level description of flowtool is provided in Appendix B Section B.5.1.

4.4.4 Protocol Transition Function Recovery. The selected inference algorithm is executed against the alphabet and sample strings created by flowtool. Because, we are using linear string models to represent samples of behavior we can select from a range of existing GI algorithm implementations. We select simple DFA for

our target automata representation. Future efforts that conduct performance analysis must further consider the choice of target automata representation so it is appropriate for performance analysis. A low level description of flowinfer processing is provided in Appendix B Section B.5.2.

4.5 *Limiting Factors*

There are practical and theoretical limitations to what we can expect to achieve. Limiting factors include our session detection technique, choice of offline analysis, and completeness of the IDEVAL data set.

4.5.1 Session Detection. Because we are attempting to recover the control flow from only samples of protocol behavior we can not exactly replicate the state transitions caused by the application stack of the distributed system. For example the state transition of the SMTP sender from INITIAL to CONN_ESTABLISHED can be inferred by the TCP connection attempt but it is not part of the application level protocol (see Figure 3.2). In fact, the initial state must be inferred by the state of the underlying TCP connection and the final state determined by understanding both the operators used internally and the state of the TCP connection.

To overcome this limitation flowtool adds TCP state operators to the alphabet and sample strings. The operators added are: TCPopen, TCPclose, TCPreset, TCP-timeout, and NIDSexit. Where TCPopen denotes the initiation of a TCP connection and TCPclose the normal termination. The TCPreset operator denotes termination of the TCP connection by a TCP RST while TCPtimeout denotes the TCP connection has timed out. Finally, the NIDSexit operator denotes that libnids has encountered the end of the trace file before the TCP connection terminated. This indicates that data capture was terminated before the data flow was completely recorded in the trace file. In other words, the trace file is incomplete.

4.5.2 Online vs. Offline Analysis. Online analysis for traffic and protocol characterization was conducted by [195, 200, 223]. Because we are using existing data sets we will concentrate on *a posteriori* offline analysis instead of *online* analysis of live execution traces. A summary of the capture file characteristics is provided in Appendix A. The weekly file characteristics are in Section A.4.2. The characteristics of the individual daily network capture are described in Section A.4.1.

4.5.3 Target Automata Representation. There are several representations that we can consider for target automata representation. Two that already provide a basis for analytical backends are CFSM and MSC as previously discussed in Section 2.8.2.2 and Section 2.8.2.1. Both the CFSM and MSC representations have characteristics which must be considered before choosing one over the other. CFSM have verification techniques including reachability and reverse reachability analysis [196]. Deadlock detection techniques are also available [96]. MSC can also be verified through process of realizability but only for bounded sizes [5]. Verifying an unbounded MSC using LTL model checking is in general undecidable [5]. While MSC might be useful for simple protocol automata the constraints on verifiability cause us to favor CFSM representations for future efforts.

Although both CFSM and MSC models provide more powerful representation we select DFA for simplicities sake in our proof of concept implementation.

4.5.4 Incomplete Data. Training data density will impact analysis [41]. We can expect only partial correctness (approximate) inference results if the input does not contain a representative sample of the protocol commands and replies. If the data set does not contain a representative sample of the protocol under investigation (i.e. the data is sparse or noisy) then the accuracy of the inference will be low. Unfortunately, the SMTP and POP3 traffic in the IDEVAL data set does not fully cover the allowed operators (Σ) in their respective specifications.

Table 4.1: POP3 Command Alphabet Weekly Overview - The IDEVAL data set does not exercise all operators allowed by the POP3 specification.

POP3 Command Alphabet Weekly Overview						
Command	Week 1	Week 2	Week 3	Week 4	Week 5	Total
STAT	255	236	395	303	321	1,510
DELE	402	400	752	409	455	2,418
USER	255	236	395	303	323	1,513
UIDL	0	0	0	0	0	0
QUIT	255	237	395	303	323	1,513
TOP	0	0	0	0	0	0
RETR	402	400	752	409	455	2,418
RSET	0	0	0	0	0	0
APOP	0	0	0	0	0	0
LIST	0	0	0	0	0	0
PASS	255	236	395	303	321	1,510
NOOP	0	0	0	0	0	0

4.5.4.1 POP3 Alphabet (Σ) Overview. For POP3 we discovered no occurrences of the following operators in the cumulative data: APOP, LIST, NOOP, RSET, TOP, UIDL. Table 4.1 summarizes the command/operator types observed in the IDEVAL data set. The POP3 specification, unlike SMTP, requires all command verbs be encoded in upper case [112]. The POP3 specification only describes two reply codes +OK and -ERR of which we observed 20,559 +OK and 60 -ERR replies. We did not parse the replies for information beyond the reply code.

4.5.4.2 SMTP Alphabet (Σ) Overview. For SMTP we discovered no occurrences of the following operators in the cumulative data: EXPN, NOOP, SEND, SOML, SAML, VRFY. The cumulative counts for discovered SMTP operators are shown in Table 4.2. It must be noted that although the SMTP has a rigid syntax the specification allows for all commands and replies to be in upper case, mixed case, or lower case [113, Section 2.4]. As shown in Table 4.2 we observed occurrences of lower case commands used by the mailbomb attack but no occurrences of mixed case commands. The SMTP Reply alphabet occurrences are summarized in Table 4.4.

Table 4.2: SMTP Cumulative Command Alphabet - The IDEVAL data set does not exercise all operators allowed by the SMTP specification.

SMTP Cumulative Command Alphabet		
Command	Count	Note
MAIL	118,029	Initiate a mail transaction
mail	1,672	Lower case MAIL
RSET	22	Abort current mail transaction.
DATA	117,899	Treat all lines as message body until data is terminated by <CR><LF>.<CR><LF>
QUIT	114,986	Server must send an OK reply and close the transmission channel
EHLO	112,131	Initiate session (extended format)
HELO	112,956	Initiate session
RCPT	186,548	Identifies an individual recipient; multiple recipients are specified by multiple occurrences
rcpt	1,670	lower case RCPT
HELP	3	Server sends helpful information to the client

4.5.5 Noisy Data. Our naïve assumption that the port used by TCP connection level traffic would indicate the type of encapsulated application level data proved wrong. Because the IDEVAL data set is designed for intrusion detection system performance evaluation it contains intentionally generated attack traffic. We consider the intentional attack traffic to be noise for our purposes. The specific types of noise that impact the alphabet and sample string creation are generated by mailbomb, tcp-reset, and SYN flood attacks. To overcome noise generated by intentional misuse we developed extrinsic filtering heuristics discussed in Section 4.6. The reader is referred to [286] for detailed descriptions of the attacks. Figure 4.2 shows the command and reply flow for an attack.

Accurate recognition of the protocol in the trace is essential to the accuracy of the k -RI and k -TSSI inference. Both algorithms are sensitive to noise if we treat all input samples as positive data. Because we are able to recognize the control flow of non-compliant traffic for SMTP and POP3 we can automatically label each sample

Table 4.3: SMTP Command Alphabet Weekly Overview - The IDEVAL data set does not exercise all operators allowed by the SMTP specification. Note that the Total column does not sum the Week columns due to incomplete TCP connection traces in the weekly pcap files.

SMTP Command Alphabet Weekly Overview						
Command	Week 1	Week 2	Week 3	Week 4	Week 5	Total
HELO	18,602	20,044	30,957	21,318	22,896	112,956
.	19,391	20,765	32,152	22,365	23,961	117,777
HELP	0	0	0	2	1	3
SAML	0	0	0	0	0	0
MAIL	19,424	20,858	32,167	22,391	24,050	118,029
mail	0	753	0	905	0	1,672
SOML	0	0	0	0	0	0
EHLO	18,244	19,701	30,420	21,263	22,570	112,131
QUIT	18,666	20,155	31,206	21774	23,124	114,900
EXPN	0	0	0	0	0	0
RSET	0	6	0	15	1	22
VRFY	0	0	0	0	0	0
SEND	0	0	0	0	0	0
RCPT	27,812	31,688	50,543	36,882	40,484	186,548
rcpt	0	753	0	905	0	1,670
DATA	19,424	20,801	32,167	22,383	23,985	117,899
NOOP	0	0	0	0	0	0

Table 4.4: SMTP Reply Alphabet Summary.

SMTP Reply Alphabet Summary		
Reply	Count	Note
220	126,266	Indicates beginning of session
250	565,331	Indicates last operation completed successfully
221	104,798	Service closing transmission channel
354	122,396	Start mail input; end with <CR><LF> . <CR><LF>
421	416	Service not available, closing transmission channel
451	209	Requested action aborted: local error in processing
500	107,073	Syntax error, command unrecognized
503	3	Bad sequence of commands
551	346	User not local; please try forward-path
552	2	Requested mail action aborted: exceeded storage allocation

Table 4.5: SMTP Reply Alphabet Weekly Overview.

SMTP Reply Alphabet Weekly Overview						
Reply	Week 1	Week 2	Week 3	Week 4	Week 5	Total
551	0	58	0	63	225	346
503	0	0	0	2	1	3
552	0	0	0	1	1	2
421	0	0	0	215	201	416
220	19,424	22,865	32,170	25,096	27,573	126,266
221	15,931	19,828	27,345	21,607	20,087	104,798
354	19,424	22,800	32,167	24,881	23,985	122,396
451	0	0	0	106	103	209
500	17,422	18,887	29,208	20,207	21,416	107,073
250	88,118	101,824	149,971	113,555	114,504	565,331

as positive or negative. This allows us to ignore the negative samples during control flow inference.

4.5.6 Connection Level Protocol Stack. The libnids library which we use to handle TCP connection level packet reassembly and defragmentation is an additional limiting factor. The library interprets TCP connection communication using a modified TCP/IP protocol stack from the Linux version 2.0.x kernel [275]. This means that the application level data presented to our flowtool will be ordered in the same manner. This problem is partially exposed by non-SMTP traffic on the SMTP port during Weeks 4 and Weeks 5 of the IDEVAL data set.

4.6 *Extrinsic Heuristics for Noise Filtering*

We concentrated first on direct naïve implementation of our format extraction algorithms followed by incremental refinement. Initial pre-processing runs contained noise caused by intentional misuse of the protocols under consideration. We used the output of the early runs to develop the filtering mechanism that automatically labeled noise sample strings as negative samples. The following criteria are used to label a sample as negative (noise):

Early Termination If the TCP connection terminates without application level protocol session termination the sample is marked negative. This means any sample ending with TCPreset, TCPTimeout, or NIDSexit is marked negative.

No Application Data An empty TCP connection without application level traffic, that is, a TCPopen followed immediately by TCPclose.

Asynchronous Command/Reply If the composite sample indicates asynchronous communication the sample is marked negative. Asynchronous communication is detected when a command follows a command, a reply follows a reply, or the sample contains only replies or commands.

Table 4.6: Buffer overflow attacks on SMTP server at 172.16.114.50 - The victim server is described as an internal host named marx.eyrie.af.mil running Redhat 4.2 kernel 2.0.27 [286]. Details for the attacking hosts, 152.204.232 and 202.49.244.10, are not provided with the IDEVAL data set.

Type	Source IP:port	Start	End	Sample
6	152.204.232.193:1941	923693227.228408	923693238.320118	TCPopen 220 250 250 250 503 HELP 500 221 TCPclose
32	202.49.244.10:1027	922715292.597701	922715294.666466	TCPopen 220 250 250 503 HELP 500 221 TCPclose
32	202.49.244.10:1027	922715290.284924	922715292.352719	TCPopen 220 250 250 503 HELP 500 221 TCPclose

Early termination was observed in 6,754 of 126,545 samples for SMTP traffic and 8,192 of 10,960 samples in POP3 traffic. As shown in Table A.5 4,432 of the 6,754 SMTP early terminations were observed during week 5 of the IDEVAL data set. A significant portion, 8,090, of the 8,192 POP3 early terminations are caused by TCP resets during week 5 (See Table A.10 in Appendix A).

An asynchronous sample is generated by the mailbomb attack (sample Type-41 in Table A.9): `TCPopen 220 mail 250 250 354 250 221 TCPclose`. The mailbomb SMTP communications are recognized as asynchronous because it terminates commands with `<CR>` instead of a standards compliant `<CR><LF>`. The SMTP servers are able to parse the non-compliant communication and pass back compliant replies that are detected by flowtool.

Another example of asynchronous behavior is generate by a buffer overflow attack, shown in Figure 4.2. Wireshark shows more detail than our flowtool because it interprets `<CR>` terminated commands. The protocol parsers in flowtool only recognize specification compliant `<CR><LF>` terminated commands. The attack generates one sample of Type-6 and two samples of Type-32 during Week 4. The three samples are summarized in Table 4.6.

While the SMTP specification does not require synchronous operation it does require synchronous communication. Reply codes indicate that processing is underway and every command must generate exactly one reply [113, Section 4.2]. Asynchronous

operation is specifically permitted during session termination where the server can send a 421 reply asynchronously after receiving a QUIT [113, Section 3.9].

Like SMTP, the POP3 specification requires synchronous communication. The POP3 specification requires an -ERR response to any unrecognized or invalid command and allows the server to automatically terminate a session after 10 minutes of inactivity [112].

4.7 Inference Accuracy

The hypothesis automata might over-restrict or over-generalize the actual target automaton of the system under consideration. If the hypothesis automata is over-restrictive it will not contain states and transitions that are necessary to accurately represent the target automaton. If the hypothesis automata over-generalizes it will contain states and transitions that are not necessary to minimally represent the target automaton. To accurately quantify the over-restrictiveness or over-generalization of the inference algorithm we must know a priori the actual automaton of the protocol under consideration. This is without regard to the performance characteristics of the target automaton and language class membership.

One option is to synthesize an approximate target automaton directly from the protocol specification. Unfortunately, the specifications for the specific protocols we are considering, SMTP and POP3, are provided in Request For Comment (RFC) documents as English language descriptions of the control flow and Augmented BNF descriptions of the operator formats. This is problematic because the English language descriptions of control flow are open to interpretation. Additionally, in general it is undecidable if a context-free grammar is regular [180, Section 4]. While a regular language (Type-3) is representable as a context-free language (Type-2) the inverse does not hold. Furthermore, transforming a context-free grammar that generates a regular language into a FSA accepting the same language is, in general, unsolvable [180, Section 4].

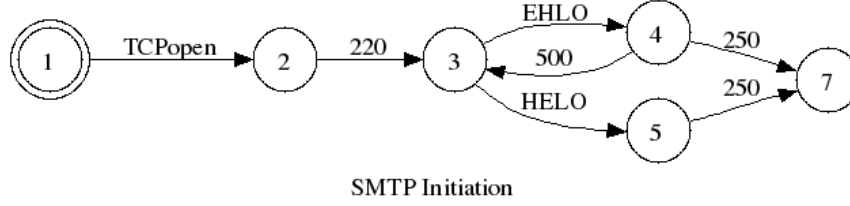


Figure 4.3: SMTP Session Initiation - SMTP session initiation starts with the TCP connection from the client to the server. The server replies with 220 then the client attempts a EHLO and if it fails HELO.

Because the protocols we are considering are described in English language (control portion) and context-free Augmented BNF (data portion) we manually generate a specification “compliant” DFA representation of the subset of the specification commands exercised by the IDEVAL data set for comparison purposes. The choice of compliance instead of conformance is intentional. The widely accepted Internet protocols described in RFC documents, unlike ISO OSI protocols, do not have a standards body that provides test suites or other conformance measurement methodologies. We limit our manually generated automata to the happy path of each protocol. That is, we do not include all possible error conditions from each state only the results of successful commands.

While the complete session for both protocols is encapsulated in a single TCP connection they do provide for session initiation, transaction, and session termination stages. The separate stages for SMTP are shown in Figure 4.3, Figure 4.4 and Figure 4.5. Our target automaton for SMTP has 19 states, 21 edges, 1 initial state, and 1 final state. The separate stages for POP3 are shown in Figure 4.6, Figure 4.7 and Figure 4.8. Our target automaton for POP3 has 15 states, 15 edges, 1 initial state, and 1 final state.

4.7.1 Inferred POP3 Control Flow. Figures for the automaton generated for POP3 by k -RI and k -TSSI inference for k values of 1, 2 and 3 are shown in Appendix C.

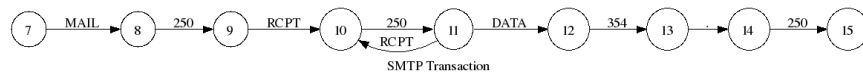


Figure 4.4: SMTP Session Transaction - A transaction stage is started when the client sends a MAIL command followed by one or more RCPT and then a DATA command terminated with a period on a line by itself.

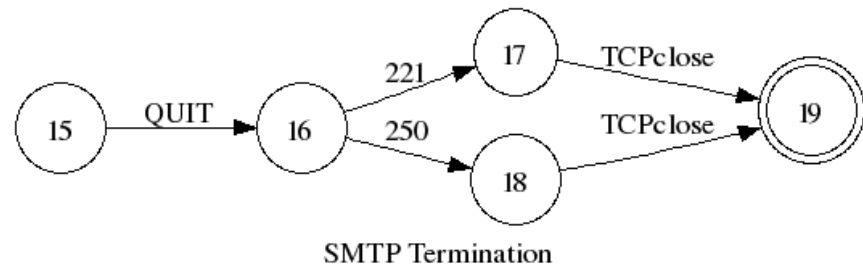


Figure 4.5: SMTP Session Termination - Session termination is initiated when the client sends a QUIT command to which the server replies with a 250 or 221 then finally the TCP connection should be closed.



Figure 4.6: POP3 Session Initiation - POP3 session initiation starts with the TCP connection from the client to the server. The server replies with +OK then the client attempts authentication with USER then PASS commands.

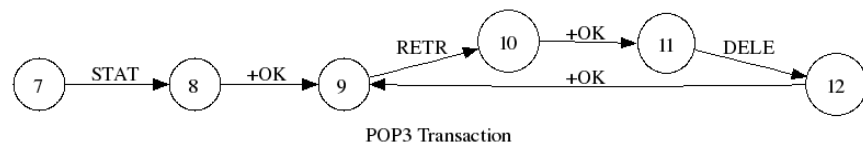


Figure 4.7: POP3 Session Transaction - After authentication the transaction stage starts which allows LIST, RECV followed by DELT, and STAT commands.

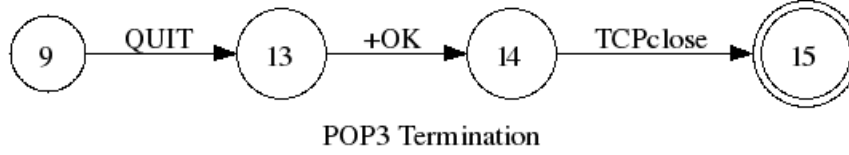


Figure 4.8: POP3 Session Termination - The session ends when the client sends a QUIT command and the server replies with +OK and finally the TCP connection is closed.

Table 4.7: k -RI POP3 Composite Automaton Filtered.

k-RI POP3 Composite Automaton Filtered				
k	States	Edges	Initial	Terminal
Target	15	15	1	1
PTA	163	162	1	18
1	15	15	1	1
2	16	17	1	1
3	18	19	1	1
4	20	21	1	1
5	22	22	1	2
6	23	24	1	2
7	25	26	1	2
8	27	28	1	2
9	29	29	1	3
10	30	31	1	3

The k -RI, with $k = 1$, inference produced an automaton that exactly matches the target automaton for the subset of the protocol exercised in the IDEVAL data set for both POP3. The k -RI inference over-generalized the target automaton at k values of 2, 3, 4, and 5. The k -TSSI inference over-restricted the inference at $k = 1$, and was equivalent to the k -RI inference for k values of 2, 3, 4, and 5. The number of states and edges for POP3 k -RI inference is shown in Table 4.7 and k -TSSI inference in Table 4.8.

4.7.2 Inferred SMTP Control Flow. Ambiguities in the specification are exhibited in the inferred control flow. The SMTP RFC allows two different replies to a QUIT command. In [113, Section 4.1.1.10] the specification states: “This com-

Table 4.8: k -TSSI POP3 Composite Automaton Filtered.

k-TSSI POP3 Composite Automaton Filtered				
k	States	Edges	Initial	Terminal
Target	15	15	1	1
1	10	15	1	1
2	16	17	1	1
3	18	19	1	1
4	20	21	1	1
5	22	22	1	2
6	23	24	1	2
7	25	26	1	2
8	27	28	1	2
9	29	29	1	3
10	30	31	1	3

Table 4.9: k -RI POP3 Composite Automaton Unfiltered.

k-RI POP3 Composite Automaton Unfiltered				
k	States	Edges	Initial	Terminal
Target	15	15	1	1
PTA	205	204	1	30
1	47	55	1	4
2	56	59	1	11
3	60	61	1	13
4	62	63	1	13
5	64	64	1	14
6	65	66	1	14
7	67	68	1	14
8	69	70	1	14
9	71	71	1	15
10	72	73	1	15

Table 4.10: k -TSSI POP3 Composite Automaton Unfiltered.

k-TSSI POP3 Composite Automaton Unfiltered				
k	States	Edges	Initial	Terminal
Target	15	15	1	1
1	14	31	1	4
2	32	38	1	11
3	39	41	1	13
4	42	44	1	13
5	45	46	1	14
6	47	49	1	14
7	50	52	1	14
8	53	55	1	14
9	56	57	1	15
10	58	60	1	15

mand specifies that the receiver MUST send an OK reply, and then close the transmission channel“. In [113, Section 4.2.2] 221 is defined as: 221 <domain> **Service closing transmission channel**. A three digit reply starting with 2 indicates positive complete, the second digit 2 indicates a transmission channel and 5 indicates status of the receiver mail system, and the third digit is used to indicate finer grain answers [113, Section 4.2.1]. The lack of clarity in the specification is reflected in the data set. The server named hume with IP number 172.16.112.100 replies to QUIT commands with 250 while others reply with 221.

The number of states and edges for SMTP k -RI inference is shown in Table 4.11 and k -TSSI inference in Table 4.12.

4.8 Sensitivity to Noise

The selected inference algorithms are directly sensitive to noise. Any sample string created by our format recognition that is not a member of the protocol under consideration will cause over-generalization of the target automaton.

One SMTP composite sample type is of particular interest because of its impact result of k -RI inference. Type-41, with 3 occurrences contains a series of valid com-

Table 4.11: k -RI SMTP Composite Automaton Filtered.

k-RI SMTP Composite Automaton Filtered				
k	States	Edges	Initial	Terminal
Target	19	21	1	1
PTA	454	453	1	36
1	60	68	1	1
2	69	76	1	2
3	77	84	1	3
4	85	92	1	4
5	93	101	1	5
6	102	111	1	6
7	112	123	1	6
8	124	136	1	7
9	137	150	1	8
10	151	165	1	9

Table 4.12: k -TSSI SMTP Composite Automaton Filtered.

k-TSSI SMTP Composite Automaton Filtered				
k	States	Edges	Initial	Terminal
Target	19	21	1	1
1	11	18	1	1
2	19	26	1	2
3	27	36	1	4
4	37	49	1	5
5	50	65	1	7
6	66	82	1	9
7	83	97	1	13
8	98	111	1	16
9	112	124	1	19
10	125	134	1	22

Table 4.13: k -RI SMTP Composite Automaton Unfiltered.

k-RI SMTP Composite Automaton Unfiltered				
k	States	Edges	Initial	Terminal
Target	19	21	1	1
PTA	587	586	1	99
1	85	131	1	4
2	133	155	1	30
3	157	172	1	40
4	175	187	1	45
5	188	198	1	49
6	199	210	1	51
7	211	226	1	51
8	227	244	1	55
9	245	261	1	58
10	262	286	1	60

Table 4.14: k -TSSI SMTP Composite Automaton Unfiltered.

k-TSSI SMTP Composite Automaton Unfiltered				
k	States	Edges	Initial	Terminal
Target	19	21	1	1
1	28	90	1	4
2	91	115	1	30
3	116	132	1	40
4	133	146	1	45
5	147	158	1	49
6	159	170	1	51
7	171	186	1	51
8	187	204	1	55
9	205	221	1	58
10	222	247	1	60

Table 4.15: k -RI SMTP Composite Automaton Type-41 removed - k -RI inference results improve when 3 occurrences of sample Type-41 are removed.

k-RI SMTP Composite Automaton Type-41 removed				
k	States	Edges	Initial	Terminal
Target	19	21	1	1
1	19	26	1	1
2	27	33	1	2
3	34	40	1	3
4	41	47	1	4
5	48	55	1	5
6	56	64	1	6
7	65	75	1	6
8	76	87	1	7
9	88	100	1	8
10	101	114	1	9

mand/reply pairs that are reset by the RSET command (See Appendix A Table A.9). The k -RI algorithm is unable to reduce the sequence resulting in over-generalization. If we remove the 3 samples marking them as negative samples k -RI inference results improve (See Table 4.15 and Table 4.11. On the other hand, k -TSSI inference is not impacted by Type-41 samples.

4.9 Algorithm Runtimes

We executed the k -RI and k -TSSI algorithms against the composite samples to develop an approximate understanding of the runtime. The runtimes presented are specific to the IDEVAL data set and the execution environment used for analysis. They should NOT be interpreted as a general performance indicator. Both algorithms were executed 250 times against the composite samples for k values 1 to 5. The executables were compiled with GCC version 3.3.6 with optimizations enabled (-O3). The environment used was openSUSE version 10.3 running on a Intel Core 2 Duo T2500 operating at 2.0 GHz based computer. The runtimes reported are the average of 250 executions.

Table 4.16: POP3 Composite Runtimes - runtimes are in seconds.

POP3 Composite Runtimes		
k	k-RI	k-TSSI
1	0.43084	0.0258
2	0.4074	0.01652
3	0.38852	0.01792
4	0.1558	0.01828
5	0.49132	0.0196

Table 4.17: SMTP Composite Runtimes - runtimes are in seconds.

SMTP Composite Runtimes		
k	k-RI	k-TSSI
1	2.73996	0.02196
2	3.02336	0.019
3	4.20492	0.0204
4	4.52704	0.02156
5	4.7618	0.0226

4.10 Chapter Summary

We presented a hybrid application of state-of-practice format recognition and grammatical inference of protocol control flow which could support the use of formal techniques to identify vulnerabilities in the specification, implementation, and deployed configuration of network protocols. We outlined our experimental design and detailed the results of format recovery and control flow recovery for POP3 and SMTP protocol traffic from the IDEVAL 1999 data set.

V. Analysis and Results

Dynamic protocol reverse engineering is a challenging problem that is unlikely to yield significant progress without research across a broad multi-disciplinary range of topics. Here we present experimental results of our proof of concept implementation and our conclusions. Finally we propose areas for future research.

5.1 Conclusions

While we have demonstrated the applicability of two grammatical inference algorithms for two specific protocols we have not established generality of the approach.

5.1.1 Experimental Results. The k -RI algorithm provided accurate inference of POP3 control flow with $k = 1$ on filtered data. The k -RI algorithm approximated our target automaton for SMTP with $k = 1$ when we removed samples that contained operators not included in our target automaton. k -TSSI over-restricted POP3 traffic with $k = 1$ and overgeneralized for other values of k . k -TSSI also over-restricted SMTP traffic for $k = 1$ and overgeneralized for other values of k .

5.1.2 Investigative Questions Answered. The focus of this research was the evaluation of existing Grammatical Inference algorithms for the dynamic protocol reverse engineering domain. We examined the following questions with the following results:

[IQ1] What information is necessary to reverse engineer the control portion of application layer protocols from data flows?

A network trace collection architecture must be constructed that is able to accurately record traces of the protocol traffic without losing samples. Next, we must have a means to reconstruct the application session. Finally, we must have access to the format of protocol operators or be able to derive the operator format.

[IQ2] Given the proven [7,95] difficulty of inferring finite automata from positive samples only, are there GI approaches that are appropriate for reverse engineering

automata representations of the control portion of application layer protocols from data flows?

For the specific protocols we examined the answer is yes within limits of data completeness. Both k -RI and k -TSSI inference were able to infer control flow that fit the target automata for POP3 and closely approximated the target automata for SMTP data observed in the IDEVAL data set. Unfortunately, the IDEVAL data set does not completely exercise either protocol resulting in incomplete automata. Finally, we have not established the generality of the approach and must leave this as an open question.

5.2 *Future Work*

This thesis presented a grammatical inference approach to reverse engineering models of protocol control flow from network traces. This is an initial step in generating tactical cyber weapons that target computer network systems. Future research could evaluate the following areas:

1. Model recovery of other classes of protocols such as: asynchronous, binary represented, multi-connection and multi-channel protocols from network traces.
2. Model recovery of higher order automata such as context-free grammars, context-sensitive grammars.
3. Model recovery from other families of protocols. While we concentrated on a subset of application level protocols on IPv4 networks similar experimental analysis could be conducted against other classes of protocols, such as SCADA or SS7, for vulnerability assessment and generation of targeted effects.
4. Online, live, and incremental model recovery. The experimental structure evaluated in this thesis requires the full (non-incremental) construction of the sample space. The k -RI and k -TSSI algorithms evaluated could support incremental modifications.

5. Formal analysis of automatically generated models ¹. Ammons presents formal analysis of specifications automatically generated from observations of instrumented applications [6]. Dallmeier examines the discovery of normal program behavior [59]. Bishop [30] studied automated specification discovery at the packet level of granularity. It could be beneficial to examine automatically created protocol specifications for implementation issues that allow deliberately crafted packets that lead a protocol parser to conditions that are unexpected.
6. Consolidation of the well known grammatical inference algorithms into an open source analysis framework like Weka [274] or Rapidminer [171], or modeling framework like Ptolemy [144] might benefit the machine learning community. The Mical [213] and Learnlib [211] projects present frameworks which implement several GI algorithms. Algorithms could be gleaned from other research efforts (e.g. [46, 169]).
7. Examine other automated or semi-automated approaches to discovering protocol defects such as RCE or randomized boundary testing ².
8. Examine cryptographic protocol verification methodologies for formalisms that can be adopted to protocol reverse engineering in general. Dongxi [66], for example, proposes an automatic attack construction algorithm to discover potential attacks on cryptographic security protocols.
9. Further examine meta-heuristic techniques such as tabu search or randomized techniques like genetic algorithms for their applicability to inference of protocol control flow.
10. Construction of a publicly releaseable research data set containing contemporary network traffic. Limitations of the IDEVAL data set used in this research are discussed in Appendix A. It would be beneficial to the network research com-

¹ [67, 273] provide overviews of formal analysis.

²Commonly referred to as fuzzing [249, Chapter 14] and [175, 188].

munity as a whole to develop a data set which addresses the concerns presented by [166].

11. Formalize results of inference mechanisms. While we presented limited empirical evidence that grammatical inference algorithms could be applied to the problem domain under consideration we did not provide formal proof of the performance characteristics.
12. Examine other language models besides linear systems built from strings. [224] presents extensions of formal languages for multi-dimensional objects such as trees and graphs or Clark’s planar languages [45, 46].

5.3 Summary

Ultimately, the grammatical inference approach presented only provides information that can assist an informed human analyst in protocol reverse engineering. The analyst will still have to apply common heuristics (e.g. identifying signpost values, block structure inference, or windowed entropy). We have provided limited empirical evidence that our grammatical inference approach to dynamic protocol reverse engineering is applicable to the protocol reverse engineering problem domain. This approach to control flow recovery should be further developed to support automated analysis of inferred control flow for performance characteristics.

Appendix A. Data

This appendix describes the data set used in this thesis and provides samples of the application level protocol traces under consideration.

A.1 Natural Data Sets

The Internet Traffic Archive (ITA) is a moderated repository to support widespread access to traces of Internet network traffic, sponsored by ACM SIGCOMM [139]. A data set was previously offered by the Passive Measurement and Analysis (PMA) project of NLANR and now by CADIA provides header traces from OC3 through OC48 speeds [167]. Unfortunately, the ITA and NLANR/PAM data sets were too narrowly focused for our research efforts and did not contain application level protocol traces.

A.2 Artificial Data Sets

While the ITA and NLANR/PAM data sets draw from real world traffic we also considered the use of artificial data sets. Various authors have proposed or constructed data sets of network traces appropriate for their area of studies [58, 215]. The LARIAT system [222] was considered for generation of application level protocol traces. Regrettably, we were not able to gain access to a working LARIAT system early enough to generate appropriate data sets.

A.3 DARPA Intrusion Detection Evaluation data set

The DARPA Intrusion Detection Evaluation data set is available from the Massachusetts Institute of Technology (MIT) Lincoln Laboratory. The data set was produced by the Information Systems Technology (IST) Group of MIT Lincoln Laboratory under Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) sponsorship [286]. The data set provides examples of attacks and background traffic. More importantly for this research it provides simulation of user generated traffic of ASCII text represented single-channel POP3 and

SMTP protocol traffic. POP3 traffic involved internal users accessing external mail servers [151]. SMTP traffic was comprised of individual, group and global email messages to and from all simulated users [151]. Like Mahoney we will refer to the data set as IDEVAL [155]. IDEVAL is both publicly available and widely used in research efforts.

A.3.1 IDEVAL Data Quality. While the IDEVAL is widely used for evaluation of intrusion detection algorithms and systems there has been some concern expressed about how accurately the data set represents more contemporary TCP/IP network activity [155]. In his assessment McHugh even questions the collection methodology, attack taxonomy and low traffic rates (among other characteristics) [166]. Mahoney and Chan analyzed the data set for simulation artifacts concluding that the data set lacked real-world ranges in the packet parameters (i.e. TTL, TCP flags, TCP windows size) [155]. Furthermore the data set lacks real-world traffic crud caused by incorrect implementations of the TCP/IP protocols [119, 155].

A.3.2 IDEVAL Data Relevance. The IDEVAL network configuration does not reflect contemporary hardware, software, or operating systems. Operating systems used include MacOS, Redhat 5.0 kernel 2.0.32, Redhat 5.2 kernel 2.0.36, Solaris 2.5.1, Solaris 2.6, SunOS 4.1.4, Windows 3.1, Windows 95, and Windows NT 4.0 build 1381 Service Pack 1 [286]. None of these operating systems are currently authorized for use in DOD networks. Dialects of network protocols are manifested in implementation specific interpretations and extensions of protocol specifications. Given, the dated OS protocol stacks used to generate traffic the IDEVAL data set might not precisely reflect current network traffic. Additionally, the topology of the network, shown in Figure A.1, is not representative of contemporary networks.

A.4 Data Files

The characteristics of the data files used are summarized in A.1 and A.2. The characteristics merged data are summarized in Table A.3 and Table A.4. The infor-

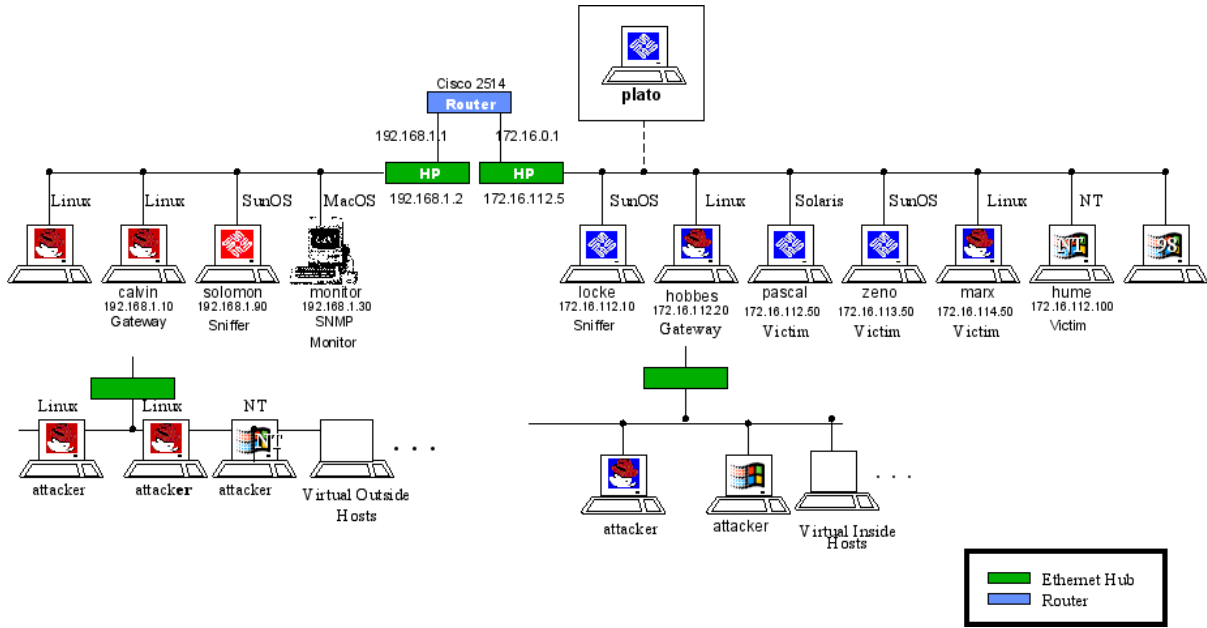


Figure A.1: IDEVAL1999 Network Topology [286].

mation was generated using the capinfos utility that accompanies Wireshark. The capinfo utility reported errors for the following files:

wk5.day2.outside.pcap An error occurred after reading 2,558,481 packets. Less data was read than expected.

wk5.day3.outside.pcap An error occurred after reading 1,385,130 packets. Less data was read than expected.

wk5.day4.outside.pcap An error occurred after reading 2,308,273 packets. Less data was read than expected.

wk5.day5.outside.pcap An error occurred after reading 2,651,589 packets. Less data was read than expected.

Additionally, the file wk2.day4.inside.pcap was not included in the data set we used.

A.4.1 Complete Data File set.

Table A.1: IDEVAL Data Files Size - Size columns are measured in bytes. The Data Rate is measured in bytes/second.

IDEVAL Data Files Size					
Name	Packets	File Size	Data Size	Data Rate (bytes/s)	Avg Size
Week 1					
wk1.day1.inside.pcap	1,492,331	341,027,537	317,150,217	4003.90	212.52
wk1.day1.outside.pcap	1,362,869	323,832,360	302,026,432	3813.47	221.61
wk1.day2.inside.pcap	1,237,119	341,401,548	321,607,620	4060.92	259.96
wk1.day2.ouside.pcap	1,157,328	325,395,277	306,878,005	3874.78	265.16
wk1.day3.inside.pcap	1,726,319	385,142,370	357,521,242	4514.33	207.10
wk1.day3.outside.pcap	1,616,713	368,776,477	342,909,045	4329.76	212.10
wk1.day4.inside.pcap	1,947,815	552,903,806	521,738,742	6588.39	267.86
wk1.day4.outside.pcap	1,807,060	517,042,040	488,129,056	6163.26	270.12
wk1.day5.inside.pcap	1,483,419	308,604,831	284,870,103	3596.96	192.04
wk1.day5.outside.pcap	1,349,635	284,774,805	263,180,621	3323.00	195.00
Week 2					
wk2.day1.inside.pcap	1,753,377	401,046,958	372,992,902	4709.84	212.73
wk2.day1.outside.pcap	1,337,777	329,322,084	307,917,628	3885.50	230.17
wk2.day2.inside.pcap	1,585,120	400,104,805	374,742,861	5462.75	236.41
wk2.day2.outside.pcap	1,454,035	375,798,588	352,534,004	5154.17	242.45
wk2.day3.inside.pcap	1,011,149	169,156,383	152,977,975	1931.66	151.29
wk2.day3.outside.pcap	888,139	145,698,730	131,488,482	1660.24	148.05
wk2.day4.outside.pcap	1,412,645	330,867,682	308,285,665	3892.62	218.23
wk2.day5.inside.pcap	1,362,422	291,511,690	269,712,914	3405.61	197.97
wk2.day5.outside.pcap	1,252,412	273,295,370	253,256,754	3197.75	202.22
Week 3					
wk3.day1.inside.extra.pcap	1,679,048	233,849,898	206,985,106	2709.36	123.28
wk3.day1.inside.pcap	2,106,744	468,024,334	434,316,406	5484.02	206.16
wk3.day1.outside.extra.pcap	1,191,358	150,014,497	130,952,745	1712.91	109.92
wk3.day1.outside.pcap	1,542,614	371,123,625	346,441,777	4374.34	224.58
wk3.day2.inside.extra.pcap	2,152,964	460,059,143	425,611,695	5387.50	197.69
wk3.day2.inside.pcap	1,831,648	414,885,615	385,579,223	4868.70	210.51
wk3.day2.outside.extra.pcap	1,822,764	403,648,042	374,483,794	4728.46	205.45
wk3.day2.outside.pcap	1,374,431	334,280,722	312,289,802	3943.11	227.21
wk3.day3.inside.pcap	1,849,753	558,991,635	529,395,563	6684.58	286.20
wk3.day3.ouside.pcap	1,760,859	540,109,859	511,936,091	6464.05	290.73
wk3.day3.outside.extra.pcap	2,453,966	766,843,295	727,579,815	9186.76	296.49
wk3.day4.inside.pcap	1,559,156	260,180,866	235,234,346	3235.69	150.87
wk3.day4.outside.pcap	1,096,660	183,158,763	165,612,179	2277.96	151.02
wk3.day5.inside.pcap	1,635,425	513,197,145	487,030,321	7939.98	297.80
Week 4					
wk4.day1.inside.pcap	1,647,573	285,359,948	258,998,756	3270.39	157.20
wk4.day1.outside.pcap	1,279,543	216,724,852	196,252,140	2478.00	153.38
wk4.day2.outside.pcap	1,309,242	301,682,860	280,734,964	3544.68	214.43
wk4.day3.inside.pcap	1,766,074	399,300,104	371,042,896	4685.55	210.09
wk4.day3.outside.pcap	1,315,032	319,141,540	298,101,004	3764.00	226.69
wk4.day4.inside.pcap	2,356,503	519,183,790	481,479,718	6080.20	204.32
wk4.day4.outside.pcap	1,635,267	399,619,424	373,455,128	4715.46	228.38
wk4.day5.inside.pcap	1,945,538	368,018,512	336,889,880	4254.06	173.16
wk4.day5.outside.pcap	1,318,345	262,141,472	241,047,928	3043.60	182.84
Week 5					
wk5.day1.inside.pcap	2,291,319	477,303,765	440,642,637	5564.14	192.31
wk5.day1.outside.pcap	1,376,598	344,257,810	322,232,218	4068.72	234.08
wk5.day2.inside.pcap	3,404,824	524,283,553	469,806,345	5932.00	137.98
wk5.day3.inside.pcap	2,087,942	491,350,468	457,943,372	5782.72	219.33
Continued on next page					

Table A.1 – continued from previous page

Name	Packets	File Size (bytes)	Data Size (bytes)	Data Rate (bytes/s)	Avg Size
wk5.day4.inside.pcap	3,201,381	826,909,800	775,687,680	9794.98	242.30
wk5.day5.inside.pcap	3,393,918	1,093,706,789	1,039,404,077	13124.91	306.25

Table A.2: IDEVAL Data Files Time - Duration is in seconds. Start and End are specified in Unix epoch time format. That is, they are specified in seconds since 00:00:00 UTC on January 1, 1970.

IDEVAL Data Files Time					
Name	Duration	Start Time	Start	End Time	End
Week 1					
wk1.day1.inside.pcap	79210.265570	Mon Mar 1 08:00:05 1999	920293205	Tue Mar 2 06:00:16 1999	920372416
wk1.day1.outside.pcap	79199.855007	Mon Mar 1 08:00:02 1999	920293202	Tue Mar 2 06:00:02 1999	920372402
wk1.day2.inside.pcap	79195.839551	Tue Mar 2 08:00:01 1999	920379601	Wed Mar 3 05:59:57 1999	920458797
wk1.day2.outside.pcap	79198.903380	Tue Mar 2 08:00:02 1999	920379602	Wed Mar 3 06:00:01 1999	920458801
wk1.day3.inside.pcap	79196.915752	Wed Mar 3 08:00:01 1999	920466001	Thu Mar 4 05:59:58 1999	920545198
wk1.day3.outside.pcap	79198.122434	Wed Mar 3 08:00:03 1999	920466003	Thu Mar 4 06:00:01 1999	920545201
wk1.day4.inside.pcap	79190.607508	Thu Mar 4 08:00:01 1999	920552401	Fri Mar 5 05:59:52 1999	920631592
wk1.day4.outside.pcap	79199.798770	Thu Mar 4 08:00:03 1999	920552403	Fri Mar 5 06:00:02 1999	920631602
wk1.day5.inside.pcap	79197.368381	Fri Mar 5 08:00:01 1999	920638801	Sat Mar 6 05:59:58 1999	920717998
wk1.day5.outside.pcap	79199.788688	Fri Mar 5 08:00:02 1999	920638802	Sat Mar 6 06:00:02 1999	920718002
Week 2					
wk2.day1.inside.pcap	79194.416159	Mon Mar 8 08:00:00 1999	920898000	Tue Mar 9 05:59:54 1999	920977194
wk2.day1.outside.pcap	79247.843302	Mon Mar 8 08:00:01 1999	920898001	Tue Mar 9 06:00:49 1999	920977249
wk2.day2.inside.pcap	68599.660226	Tue Mar 9 08:00:01 1999	920984401	Wed Mar 10 03:03:21 1999	921053001
wk2.day2.outside.pcap	68397.777830	Tue Mar 9 08:00:01 1999	920984401	Wed Mar 10 02:59:59 1999	921052799
wk2.day3.inside.pcap	79194.939151	Wed Mar 10 08:00:02 1999	921070802	Thu Mar 11 05:59:57 1999	921149997
wk2.day3.outside.pcap	79198.605427	Wed Mar 10 08:00:03 1999	921070803	Thu Mar 11 06:00:01 1999	921150001
wk2.day4.outside.pcap	79197.414198	Thu Mar 11 08:00:03 1999	921157203	Fri Mar 12 06:00:00 1999	921236400
wk2.day5.inside.pcap	79196.548757	Fri Mar 12 08:00:01 1999	921243601	Sat Mar 13 05:59:58 1999	921322798
wk2.day5.outside.pcap	79198.411013	Fri Mar 12 08:00:02 1999	921243602	Sat Mar 13 06:00:00 1999	921322800
Week 3					
wk3.day1.inside.extra.pcap	76396.316028	Mon Mar 22 08:00:02 1999	922107602	Tue Mar 23 05:13:19 1999	922183999
wk3.day1.inside.pcap	79196.697590	Mon Mar 15 08:00:01 1999	921502801	Tue Mar 16 05:59:58 1999	921581998
wk3.day1.outside.extra.pcap	76450.306697	Mon Mar 22 08:00:03 1999	922107603	Tue Mar 23 05:14:14 1999	922184054
wk3.day1.outside.pcap	79198.682477	Mon Mar 15 08:00:02 1999	921502802	Tue Mar 16 06:00:00 1999	921582000
wk3.day2.inside.extra.pcap	78999.815637	Tue Mar 23 08:00:02 1999	922194002	Wed Mar 24 05:56:42 1999	922273002
wk3.day2.inside.pcap	79195.474873	Tue Mar 16 08:00:00 1999	921589200	Wed Mar 17 05:59:55 1999	921668395
wk3.day2.outside.extra.pcap	79197.824660	Tue Mar 23 08:00:00 1999	922194000	Wed Mar 24 05:59:58 1999	922273198
wk3.day2.outside.pcap	79198.800883	Tue Mar 16 08:00:01 1999	921589201	Wed Mar 17 06:00:00 1999	921668400
wk3.day3.inside.pcap	79196.540665	Wed Mar 17 08:00:01 1999	921675601	Thu Mar 18 05:59:58 1999	921754798
wk3.day3.outside.pcap	79197.391311	Wed Mar 17 08:00:03 1999	921675603	Thu Mar 18 06:00:00 1999	921754800
wk3.day3.outside.extra.pcap	79198.759438	Wed Mar 24 08:00:01 1999	922280401	Thu Mar 25 06:00:00 1999	922359600
wk3.day4.inside.pcap	72699.913792	Thu Mar 18 08:00:03 1999	921762003	Fri Mar 19 04:11:42 1999	921834702
wk3.day4.outside.pcap	72702.007896	Thu Mar 18 08:00:02 1999	921762002	Fri Mar 19 04:11:44 1999	921834704
wk3.day5.inside.pcap	61338.967582	Fri Mar 19 08:00:02 1999	921848402	Sat Mar 20 01:02:21 1999	921909741
Week 4					
wk4.day1.inside.pcap	79195.171194	Mon Mar 29 08:00:02 1999	922712402	Tue Mar 30 05:59:57 1999	922791597
wk4.day1.outside.pcap	79197.929511	Mon Mar 29 08:00:03 1999	922712403	Tue Mar 30 06:00:01 1999	922791601
wk4.day2.outside.pcap	79198.923458	Tue Mar 30 08:00:02 1999	922798802	Wed Mar 31 06:00:01 1999	922878001
wk4.day3.inside.pcap	79188.757556	Wed Mar 31 08:00:09 1999	922885209	Thu Apr 1 05:59:57 1999	922964397

Continued on next page

Table A.2 – continued from previous page

Name	Duration	Start Time	Start	End Time	End
wk4.day3.outside.pcap	79197.915230	Wed Mar 31 08:00:02 1999	922885202	Thu Apr 1 06:00:00 1999	922964400
wk4.day4.inside.pcap	79188.179768	Thu Apr 1 08:00:01 1999	922971601	Fri Apr 2 05:59:49 1999	923050789
wk4.day4.outside.pcap	79198.095408	Thu Apr 1 08:00:03 1999	922971603	Fri Apr 2 06:00:01 1999	923050801
wk4.day5.inside.pcap	79192.491900	Fri Apr 2 08:00:00 1999	923058000	Sat Apr 3 05:59:53 1999	923137193
wk4.day5.outside.pcap	79198.366773	Fri Apr 2 08:00:01 1999	923058001	Sat Apr 3 06:00:00 1999	923137200
Week 5					
wk5.day1.inside.pcap	79193.374094	Mon Apr 5 08:00:02 1999	923313602	Tue Apr 6 05:59:56 1999	923392796
wk5.day1.outside.pcap	79197.375906	Mon Apr 5 08:00:03 1999	923313603	Tue Apr 6 06:00:00 1999	923392800
wk5.day2.inside.pcap	79198.608846	Tue Apr 6 08:00:00 1999	923400000	Wed Apr 7 05:59:58 1999	923479198
wk5.day3.inside.pcap	79191.650256	Wed Apr 7 08:00:00 1999	923486400	Thu Apr 8 05:59:52 1999	923565592
wk5.day4.inside.pcap	79192.349066	Thu Apr 8 08:00:00 1999	923572800	Fri Apr 9 05:59:53 1999	923651993
wk5.day5.inside.pcap	79193.248408	Fri Apr 9 08:00:04 1999	923659204	Sat Apr 10 05:59:58 1999	923738398

A.4.2 Merged Data File set. The choice of offline analysis allowed us to filter the raw IDEVAL data set to include only the POP3 and SMTP protocol traffic. We used the command line interface to Wireshark, called tshark, to filter out the protocols under consideration from each of the daily capture files. For example:

```
tshark -R "tcp.port eq 25 or tcp.port eq 110"
-r <input file name>
-w <output file name>
```

extracts SMTP and POP3 traffic from the <input file name> and writes it to the <output file name>.

Next we used mergecap, a command line utility also distributed with Wireshark, to merge the filtered data into weekly summary files.

```
mergcap -F libpcap -w wk1.pcap 'find . -iname "wk1.*.filtered.pcap"'
```

Finally, we merged the weekly files into a cumulative trace file named total.pcap.

The characteristics merged data are summarized in Table A.3 and Table A.4. The pre-processing workflow is shown in Figure A.2. Note that the cumulative alphabet is input into weekly sample extractions to ensure that the sample strings produced use a common alphabet.

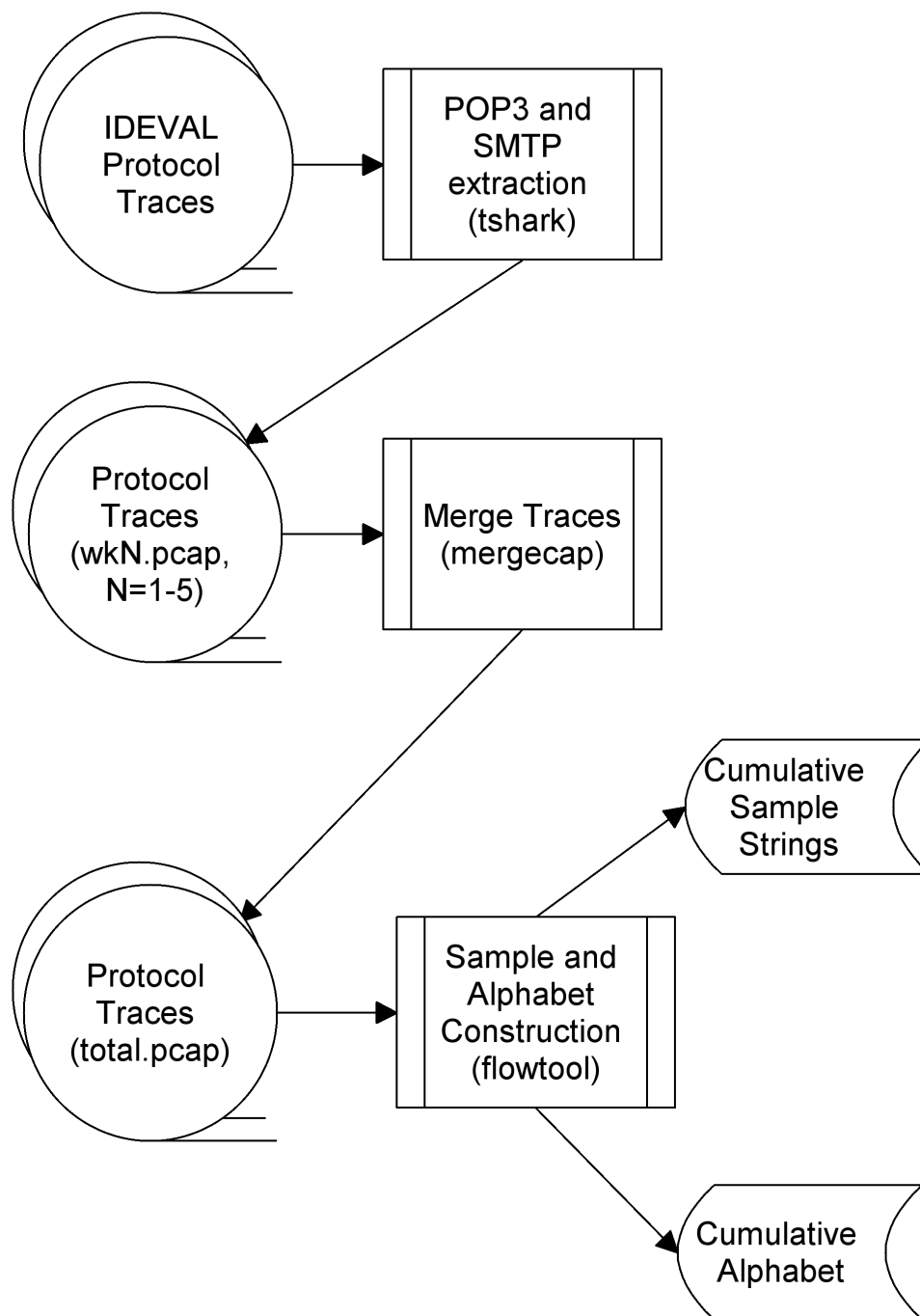


Figure A.2: Experimental Architecture Pre-Processing.

Table A.3: Merged Data Files Size - Size columns are measured in bytes. The Data Rate is measured in bytes/second.

Merged Data Files Size					
Name	Packets	File Size	Data Size	Data Rate (bytes/s)	Avg Size
wk1.pcap	619,215	107,546,006	97,638,542	230.02	157.68
wk2.pcap	640,110	107,405,071	97,163,287	228.85	151.79
wk3.pcap	899,064	156,829,234	142,444,186	170.55	158.44
wk4.pcap	708,272	124,405,656	113,073,280	266.94	159.65
wk5.pcap	897,033	150,206,030	135,853,478	328.73	151.45

Table A.4: Merged Data Files Time - Duration is in seconds. Start and End are specified in Unix epoch time format. That is, they are specified in seconds since 00:00:00 UTC on January 1, 1970.

Merged Data Files Time					
Name	Duration	Start Time	Start	End Time	End
wk1.pcap	424481.648063	Mon Mar 1 08:00:40 1999	920293240	Sat Mar 6 05:55:22 1999	920717722
wk2.pcap	424565.061365	Mon Mar 8 08:00:04 1999	920898004	Sat Mar 13 05:56:09 1999	921322569
wk3.pcap	835181.388736	Mon Mar 15 08:00:18 1999	921502818	Wed Mar 24 23:59:59 1999	922337999
wk4.pcap	423583.955343	Mon Mar 29 08:00:02 1999	922712402	Sat Apr 3 05:39:46 1999	923135986
wk5.pcap	413264.972580	Mon Apr 5 08:00:31 1999	923313631	Sat Apr 10 02:48:16 1999	923726896

Table A.5: SMTP TCP Connection Summary - Note that like POP3 the weekly pcap files do not correctly contain the entire TCP connection for several of the SMTP connections. The Total column is NOT the sum of the row.

SMTP TCP Connection Summary							
TCP Operation	Week 1	Week 2	Week 3	Week 4	Week 5	Total	Percent
TCPOpen	19,424	22,868	32,180	25,155	27,783	126,545	100
TCPclose	18,687	22,172	31,212	24,398	23,351	119,791	25.25
TCPreset	5	9	60	253	3,730	3,957	73.81
TCPTimeout	0	0	146	0	424	2,519	0.07
NIDSexit	732	687	762	504	278	278	0.85
Total termination conditions: 126,545							

A.5 SMTP Sample Data

Table A.7: Data Summary: SMTP Command Alphabet total.pcap

Data Summary: SMTP Command Alphabet total.pcap				
Type	Count	Percent	Label	Sample
	1	0.000790233	+	TCPOpen EHLO MAIL RCPT DATA . TCPclose
2	1	0.000790233	+	TCPOpen EHLO MAIL RCPT(31) QUIT TCPclose
3	1	0.000790233	+	TCPOpen HELO MAIL RCPT QUIT TCPclose
4	1	0.000790233	+	TCPOpen HELO MAIL RCPT(6) DATA . QUIT TCPclose
5	1	0.000790233	+	TCPOpen HELO MAIL RCPT(7) DATA . QUIT TCPclose
6	1	0.000790233	-	TCPOpen EHLO HELO MAIL RCPT TCPTIMEOUT
7	1	0.000790233	-	TCPOpen EHLO HELO MAIL RCPT(11) DATA TCPTIMEOUT
8	1	0.000790233	-	TCPOpen EHLO HELO MAIL RCPT(2) DATA . NIDSEXIT
9	1	0.000790233	-	TCPOpen EHLO HELO MAIL RCPT(20) TCPTIMEOUT
10	1	0.000790233	-	TCPOpen EHLO HELO TCPTIMEOUT
11	1	0.000790233	-	TCPOpen EHLO MAIL RCPT DATA . TCPreset
12	1	0.000790233	-	TCPOpen EHLO MAIL RCPT DATA . TCPTIMEOUT
13	1	0.000790233	-	TCPOpen EHLO MAIL RCPT DATA NIDSEXIT
14	1	0.000790233	-	TCPOpen EHLO MAIL TCPTIMEOUT
15	1	0.000790233	-	TCPOpen EHLO TCPTIMEOUT
16	1	0.000790233	-	TCPOpen HELO MAIL RCPT DATA . QUIT NIDSEXIT
17	1	0.000790233	-	TCPOpen HELO MAIL TCPTIMEOUT
18	1	0.000790233	-	TCPOpen HELO TCPTIMEOUT
19	2	0.00158047	+	TCPOpen EHLO HELO MAIL RCPT(7) DATA . QUIT TCPclose
20	2	0.00158047	+	TCPOpen EHLO MAIL RCPT(2) QUIT TCPclose
21	2	0.00158047	+	TCPOpen EHLO MAIL RCPT(4) QUIT TCPclose
22	2	0.00158047	-	TCPOpen EHLO HELO MAIL RCPT DATA . QUIT TCPTIMEOUT
23	2	0.00158047	-	TCPOpen EHLO HELO MAIL RCPT DATA TCPreset
24	2	0.00158047	-	TCPOpen EHLO HELO MAIL RCPT(5) DATA TCPclose
25	2	0.00158047	-	TCPOpen EHLO MAIL RCPT(7) DATA TCPclose
26	2	0.00158047	-	TCPOpen HELO MAIL RCPT DATA . QUIT TCPTIMEOUT
27	2	0.00158047	-	TCPOpen mail TCPclose
28	3	0.0023707	+	TCPOpen EHLO HELO MAIL RCPT DATA . RSET MAIL RCPT DATA . RSET MAIL RCPT DATA . RSET MAIL RCPT DATA . RSET MAIL RCPT DATA . QUIT TCPclose
29	3	0.0023707	-	TCPOpen EHLO HELO MAIL RCPT DATA NIDSEXIT
30	3	0.0023707	-	TCPOpen EHLO HELO MAIL RCPT(2) DATA . TCPTIMEOUT
31	3	0.0023707	-	TCPOpen EHLO HELO MAIL RCPT(30) DATA . TCPTIMEOUT
32	3	0.0023707	-	TCPOpen HELP TCPclose
Continued on next page				

Table A.7 – continued from previous page

Type	Count	Percent	Label	Sample
33	4	0.00316093	+	TCPopen EHLO MAIL RCPT(3) QUIT TCPclose
34	4	0.00316093	+	TCPopen HELO MAIL RCPT(4) DATA . QUIT TCPclose
35	4	0.00316093	-	TCPopen EHLO HELO MAIL RCPT(3) DATA TCPclose
36	4	0.00316093	-	TCPopen EHLO MAIL RCPT DATA TCPclose
37	5	0.00395116	+	TCPopen HELO MAIL RCPT(5) DATA . QUIT TCPclose
38	6	0.0047414	+	TCPopen EHLO MAIL RCPT(30) QUIT TCPclose
39	6	0.0047414	-	TCPopen EHLO HELO MAIL RCPT DATA . NIDSexit
40	6	0.0047414	-	TCPopen EHLO HELO MAIL RCPT(30) DATA TCPtimeout
41	6	0.0047414	-	TCPopen QUIT TCPclose
42	7	0.00553163	+	TCPopen EHLO MAIL RCPT RSET QUIT TCPclose
43	10	0.00790233	-	TCPopen EHLO HELO MAIL RCPT(2) DATA TCPclose
44	20	0.0158047	+	TCPopen EHLO HELO MAIL RCPT(31) DATA . QUIT TCPclose
45	21	0.0165949	-	TCPopen EHLO HELO MAIL RCPT DATA . QUIT TCPreset
46	23	0.0181754	-	TCPopen EHLO HELO MAIL RCPT DATA TCPtimeout
47	26	0.0205461	-	TCPopen TCPtimeout
48	33	0.0260777	-	TCPopen NIDSexit
49	36	0.0284484	+	TCPopen HELO MAIL RCPT(3) DATA . QUIT TCPclose
50	57	0.0450433	+	TCPopen EHLO MAIL RCPT(4) DATA . QUIT TCPclose
51	58	0.0458335	-	TCPopen EHLO HELO MAIL RCPT DATA . TCPreset
52	64	0.0505749	-	TCPopen EHLO HELO MAIL RCPT DATA TCPclose
53	65	0.0513651	+	TCPopen EHLO MAIL RCPT(3) DATA . QUIT TCPclose
54	93	0.0734916	+	TCPopen EHLO MAIL RCPT(7) DATA . QUIT TCPclose
55	103	0.081394	+	TCPopen EHLO MAIL RCPT QUIT TCPclose
56	112	0.0885061	+	TCPopen EHLO HELO MAIL RCPT(6) DATA . QUIT TCPclose
57	185	0.146193	+	TCPopen EHLO HELO MAIL RCPT(5) DATA . QUIT TCPclose
58	189	0.149354	+	TCPopen EHLO HELO MAIL RCPT(11) DATA . QUIT TCPclose
59	190	0.150144	+	TCPopen EHLO MAIL RCPT(10) DATA . QUIT TCPclose
60	199	0.157256	+	TCPopen EHLO MAIL RCPT(24) DATA . QUIT TCPclose
61	212	0.167529	+	TCPopen HELO MAIL RCPT(2) DATA . QUIT TCPclose
62	233	0.184124	-	TCPopen HELO MAIL RCPT DATA . NIDSexit
63	242	0.191236	-	TCPopen HELO MAIL RCPT DATA . TCPreset
64	433	0.342171	+	TCPopen EHLO MAIL RCPT(2) DATA . QUIT TCPclose
65	449	0.354814	-	TCPopen EHLO HELO MAIL RCPT DATA . TCPtimeout
66	560	0.44253	+	TCPopen EHLO HELO MAIL RCPT(4) DATA . QUIT TCPclose
67	1494	1.18061	+	TCPopen EHLO HELO MAIL RCPT(30) DATA . QUIT TCPclose
68	1670	1.31969	-	TCPopen mail rcpt TCPclose
69	1933	1.52752	+	TCPopen EHLO HELO MAIL RCPT(3) DATA . QUIT TCPclose
70	1996	1.5773	-	TCPopen HELO MAIL RCPT DATA . TCPtimeout
71	3150	2.48923	+	TCPopen HELO MAIL RCPT DATA . QUIT TCPclose
72	3155	2.49318	-	TCPopen TCPclose
73	3633	2.87092	-	TCPopen TCPreset
74	3887	3.07163	+	TCPopen EHLO MAIL RCPT DATA . QUIT TCPclose
75	7389	5.83903	+	TCPopen EHLO HELO MAIL RCPT(2) DATA . QUIT TCPclose
76	94522	74.6944	+	TCPopen EHLO HELO MAIL RCPT DATA . QUIT TCPclose
76 Start with: TCPopen				
45 End with: TCPclose (119,791 samples)				
6 End with: TCPreset (3,957 samples)				
18 End with: TCPtimeout (2,519 samples)				
7 End with: NIDSexit (278 samples)				
34 Positive sample Types. 114,869 Positive samples.				
42 Negative sample Types. 11,676 Negative samples.				
126,545 Total Samples				
76 Unique Types				

Table A.8: Data Summary: SMTP Reply Alphabet total.pcar

Data Summary: SMTP Reply Alphabet total.pcap				
Type	Count	Percent	Label	Sample
1	1	0.000790233	+	TCPopen 220 250(2) 551(31) 221 TCPclose
2	1	0.000790233	+	TCPopen 220 250(3) 354 250 TCPclose
3	1	0.000790233	+	TCPopen 220 250(8) 354 250 221 TCPclose
4	1	0.000790233	-	TCPopen 220 250(3) 354 250 221 NIDSexit
5	1	0.000790233	-	TCPopen 220 250(3) 354 250 TCPreset
6	1	0.000790233	-	TCPopen 220 250(3) 354 552 NIDSexit
7	1	0.000790233	-	TCPopen 220 250(3) 354 552 TCPtimeout
8	1	0.000790233	-	TCPopen 220 250(3) 503 500 221 TCPclose
9	1	0.000790233	-	TCPopen 220 500 250 TCPtimeout
10	1	0.000790233	-	TCPopen 220 500 250(13) 354 250 221 TCPtimeout
11	1	0.000790233	-	TCPopen 220 500 250(2) TCPtimeout
12	1	0.000790233	-	TCPopen 220 500 250(21) TCPtimeout
13	1	0.000790233	-	TCPopen 220 500 250(3) 354 250(2) NIDSexit
14	1	0.000790233	-	TCPopen 220 500 250(3) TCPtimeout
15	1	0.000790233	-	TCPopen 220 500 250(32) 354 TCPtimeout
16	1	0.000790233	-	TCPopen 220 500 250(4) 354 NIDSexit
17	2	0.00158047	+	TCPopen 220 250(2) 551(2) 221 TCPclose
18	2	0.00158047	+	TCPopen 220 250(2) 551(4) 221 TCPclose
19	2	0.00158047	+	TCPopen 220 500 250(9) 354 250 221 TCPclose
20	2	0.00158047	-	TCPopen 220 250 TCPtimeout
21	2	0.00158047	-	TCPopen 220 250(2) 503 500 221 TCPclose
22	2	0.00158047	-	TCPopen 220 250(3) 354 250 221 TCPtimeout
23	2	0.00158047	-	TCPopen 220 451 TCPreset
24	2	0.00158047	-	TCPopen 220 500 250(3) 354 250 221 NIDSexit
25	3	0.0023707	+	TCPopen 220 500 250(3) 354 250(4) 354 250(4) 354 250(4) 354 250(4) 354 250(4) 354 250 221 TCPclose
26	3	0.0023707	-	TCPopen 220 500 250(3) 354 250 TCPtimeout
27	3	0.0023707	-	TCPopen 220 500 250(4) 354 250 221 TCPtimeout
28	4	0.00316093	+	TCPopen 220 250(2) 551(3) 221 TCPclose
29	5	0.00395116	+	TCPopen 220 250(7) 354 250 221 TCPclose
30	6	0.0047414	+	TCPopen 220 250(2) 551(30) 221 TCPclose
31	6	0.0047414	-	TCPopen 220 221 TCPclose
32	6	0.0047414	-	TCPopen 220 500 250(3) 354 NIDSexit
33	6	0.0047414	-	TCPopen 220 TCPclose
34	7	0.00553163	+	TCPopen 220 250(2) 551 250 221 TCPclose
35	8	0.00632186	-	TCPopen 220 500 250(32) 354 250 221 TCPtimeout
36	10	0.00790233	-	TCPopen 220 TCPtimeout
37	11	0.00869256	-	TCPopen 220 500 250(3) 354 250 221 TCPtimeout
38	18	0.0142242	-	TCPopen TCPtimeout
39	20	0.0158047	+	TCPopen 220 500 250(33) 354 250 221 TCPclose
40	22	0.0173851	-	TCPopen 220 500 250(3) 354 250 TCPreset
41	25	0.0197558	-	TCPopen 220 250(2) 354 250 TCPclose
42	33	0.0260777	+	TCPopen 220 250(3) 354 250(2) TCPclose
43	33	0.0260777	-	TCPopen NIDSexit
44	37	0.0292386	-	TCPopen 220 451 421 TCPreset
45	59	0.0466237	-	TCPopen 220 500 250(3) 354 TCPreset
46	61	0.0482042	+	TCPopen 220 250(6) 354 250 221 TCPclose
47	75	0.0592675	-	TCPopen TCPreset
48	96	0.0758623	+	TCPopen 220 250(9) 354 250 221 TCPclose
49	101	0.0798135	+	TCPopen 220 250(5) 354 250 221 TCPclose

Continued on next page

Type	Count	Percent	Label	Sample
50	104	0.0821842	+	TCPopen 220 250(2) 551 221 TCPclose
51	112	0.0885061	+	TCPopen 220 500 250(8) 354 250 221 TCPclose
52	153	0.120906	-	TCPopen TCPclose
53	170	0.13434	-	TCPopen 220 451 421 TCPclose
54	187	0.147774	+	TCPopen 220 500 250(7) 354 250 221 TCPclose
55	189	0.149354	+	TCPopen 220 500 250(13) 354 250 221 TCPclose
56	190	0.150144	+	TCPopen 220 250(12) 354 250 221 TCPclose
57	199	0.157256	+	TCPopen 220 250(26) 354 250 221 TCPclose
58	209	0.165159	-	TCPopen 220 421 TCPreset
59	233	0.184124	-	TCPopen 220 250(3) 354 NIDSexit
60	242	0.191236	-	TCPopen 220 250(3) 354 TCPreset
61	459	0.362717	-	TCPopen 220 500 250(3) 354 TCPtimeout
62	560	0.44253	+	TCPopen 220 500 250(6) 354 250 221 TCPclose
63	645	0.5097	+	TCPopen 220 250(4) 354 250 221 TCPclose
64	1494	1.18061	+	TCPopen 220 500 250(32) 354 250 221 TCPclose
65	1937	1.53068	+	TCPopen 220 500 250(5) 354 250 221 TCPclose
66	1996	1.5773	-	TCPopen 220 250(3) 354 TCPtimeout
67	3310	2.61567	-	TCPopen 220 TCPreset
68	4473	3.53471	-	TCPopen 220 250(2) 354 250 221 TCPclose
69	7008	5.53795	+	TCPopen 220 250(3) 354 250 221 TCPclose
70	7399	5.84693	+	TCPopen 220 500 250(4) 354 250 221 TCPclose
71	14633	11.5635	+	TCPopen 220 500 250(3) 354 250(2) TCPclose
72	79953	63.1815	+	TCPopen 220 500 250(3) 354 250 221 TCPclose
72 Start with: TCPopen				
38 End with: TCPclose (119,791 samples)				
9 End with: TCPreset (3,957 samples)				
17 End with: TCPtimeout (2,519 samples)				
8 End with: NIDSexit (278 samples)				
30 Positive sample Types. 114,955 Positive samples.				
42 Negative sample Types. 11,590 Negative samples.				
126,545 Total Samples				
72 Unique Types				

Data Summary: SMTP Composite Alphabet total.pcap				
Type	Count	Percent	Label	Sample
1	1	0.000790233	+	TCPopen 220 EHLO 250 MAIL 250 RCPT 250 DATA 354 . 250 TCPclose
2	1	0.000790233	+	TCPopen 220 EHLO 250 MAIL 250 RCPT 551 RCPT 221 TCPclose
3	1	0.000790233	+	TCPopen 220 HELO 250 MAIL 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
4	1	0.000790233	+	TCPopen 220 HELO 250 MAIL 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
5	1	0.000790233	+	TCPopen 220 HELO 250 MAIL 250 RCPT 551 QUIT 221 TCPclose
6	1	0.000790233	-	TCPopen 220 250(3) 503 HELP 500 221 TCPclose
Continued on next page				

Table A.9 – continued from previous page

Type	Count	Percent	Label	Sample
7	1	0.000790233	-	TCPOPEN 220 500 250(3) 354 250 221 TCPtimeout
8	1	0.000790233	-	TCPOPEN 220 EHLO 250 MAIL 250 RCPT 250 DATA 354 . 250 221 TCPtime- out
9	1	0.000790233	-	TCPOPEN 220 EHLO 250 MAIL 250 RCPT 250 DATA 354 . 250 TCPreset
10	1	0.000790233	-	TCPOPEN 220 EHLO 250 MAIL 250 RCPT 250 DATA 354 250 221 NIDSexit
11	1	0.000790233	-	TCPOPEN 220 EHLO 250 MAIL TCPtimeout
12	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT TCPtimeout
13	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 TCPreset
14	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 250(2) NIDSexit
15	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA TCPtimeout
16	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 RCPT 250 DATA 354 . NIDSexit
17	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 DATA 354 250 221 TCPtimeout
18	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . TCPtimeout
19	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT TCPtimeout
20	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT TCPtimeout
21	1	0.000790233	-	TCPOPEN 220 EHLO 500 HELO 250 TCPtimeout
22	1	0.000790233	-	TCPOPEN 220 EHLO HELO MAIL RCPT DATA . QUIT TCPtimeout
23	1	0.000790233	-	TCPOPEN 220 EHLO TCPtimeout
24	1	0.000790233	-	TCPOPEN 220 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPtimeout
25	1	0.000790233	-	TCPOPEN 220 HELO 250 MAIL 250 RCPT 250 DATA 354 . 552 QUIT NIDSexit
26	1	0.000790233	-	TCPOPEN 220 HELO 250 MAIL 250 RCPT 250 DATA 354 552 . QUIT TCPti- meout
27	1	0.000790233	-	TCPOPEN 220 HELO 250 MAIL TCPtimeout
28	1	0.000790233	-	TCPOPEN 220 HELO TCPtimeout
29	2	0.00158047	+	TCPOPEN 220 EHLO 250 MAIL 250 RCPT 551 RCPT 551 QUIT 221 TCPclose
30	2	0.00158047	+	TCPOPEN 220 EHLO 250 MAIL 250 RCPT 551 RCPT 551 RCPT 551 RCPT 551 QUIT 221 TCPclose
31	2	0.00158047	+	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
32	2	0.00158047	-	TCPOPEN 220 250(2) 503 HELP 500 221 TCPclose
33	2	0.00158047	-	TCPOPEN 220 451 TCPreset
34	2	0.00158047	-	TCPOPEN 220 EHLO 250 MAIL 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 RCPT 250 DATA 354 250 221 TCPclose
35	2	0.00158047	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 TCPtimeout
36	2	0.00158047	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 250 221 NIDSexit
37	2	0.00158047	-	TCPOPEN 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 TCPreset

Continued on next page

Table A.9 – continued from previous page

Type	Count	Percent	Label	Sample
89	1494	1.18061	+	TCPopen 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
90	1657	1.30942	-	TCPopen 220 mail 250 rcpt 250 354 250 221 TCPclose
91	1933	1.52752	+	TCPopen 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 RCPT 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
92	1996	1.5773	-	TCPopen 220 HELO 250 MAIL 250 RCPT 250 DATA 354 . TCPTIMEOUT
93	2814	2.22371	-	TCPopen 220 250(2) 354 250 221 TCPclose
94	3117	2.46316	+	TCPopen 220 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
95	3310	2.61567	-	TCPopen 220 TCPreset
96	3887	3.07163	+	TCPopen 220 EHLO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
97	7389	5.83903	+	TCPopen 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
98	14633	11.5635	+	TCPopen 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT 250 TCPclose
99	79889	63.1309	+	TCPopen 220 EHLO 500 HELO 250 MAIL 250 RCPT 250 DATA 354 . 250 QUIT 221 TCPclose
99 Start with: TCPopen				
53 End with: TCPclose (119,791 samples)				
11 End with: TCPreset (3,957 samples)				
27 End with: TCPTIMEOUT (2,519 samples)				
8 End with: NIDSExit (278 samples)				
36 Positive sample Types. 114,869 Positive samples.				
63 Negative sample Types. 11,676 Negative samples.				
126,545 Total Samples				
99 Unique Types				

Table A.10: POP3 TCP Connection Summary - Note that the weekly trace files do not include the complete TCP connection for all POP3 sessions. The Total column does NOT sum the row.

POP3 TCP Connection Summary							
TCP Operation	Week 1	Week 2	Week 3	Week 4	Week 5	Total	Percent
TCPopen	255	238	395	363	9,709	10,960	100
TCPclose	254	236	395	363	1,520	2,768	25.25
TCPreset	0	1	0	0	8,089	8,090	73.81
TCptimeout	0	0	0	0	6	8	0.07
NIDSexit	1	1	0	0	94	94	0.85
Total termination conditions: 10,960							

A.6 POP3 Sample Data

Table A.12: Data Summary: POP3 Command Alphabet total.pcap

Data Summary: POP3 Command Alphabet total.pcap				
Type	Count	Percent	Label	Sample
	1	0.00912409	+	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPclose
2	1	0.00912409	-	TCPopen QUIT TCPtimeout
3	1	0.00912409	-	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPtimeout
4	1	0.00912409	-	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPtimeout
5	2	0.0182482	+	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPclose
6	2	0.0182482	+	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPclose
7	2	0.0182482	+	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPclose
8	2	0.0182482	+	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPclose
9	2	0.0182482	+	TCPopen USER PASS STAT RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPclose
10	2	0.0182482	+	TCPopen USER PASS STAT RETR DELE QUIT TCPclose

Continued on next page

[illegible]

Data Summary: POP3 Reply Alphabet total.pcap				
Type	Count	Percent	Label	Sample
1	1	0.00912409	+	TCPopen OK(41) TCPclose
2	1	0.00912409	-	TCPopen OK(2) TCPclose
3	1	0.00912409	-	TCPopen OK(2) TCPtimeout
4	2	0.0182482	+	TCPopen OK(27) TCPclose
5	2	0.0182482	+	TCPopen OK(29) TCPclose
6	2	0.0182482	+	TCPopen OK(33) TCPclose
7	2	0.0182482	+	TCPopen OK(37) TCPclose
Continued on next page				

Type	Count	Percent	Label	Sample
8	2	0.0182482	+	TCPopen OK(43) TCPclose
9	2	0.0182482	+	TCPopen OK(49) TCPclose
10	2	0.0182482	+	TCPopen OK(55) TCPclose
11	2	0.0182482	-	TCPopen OK(15) TCPtimeout
12	5	0.0456204	+	TCPopen OK(23) TCPclose
13	5	0.0456204	-	TCPopen TCPtimeout
14	9	0.0821168	+	TCPopen OK(21) TCPclose
15	15	0.136861	-	TCPopen TCPreset
16	25	0.228102	+	TCPopen OK(19) TCPclose
17	32	0.291971	+	TCPopen OK(17) TCPclose
18	47	0.428832	+	TCPopen OK(15) TCPclose
19	60	0.547445	-	TCPopen OK(2) ERR TCPclose
20	69	0.629562	+	TCPopen OK(13) TCPclose
21	94	0.857664	-	TCPopen NIDSexit
22	136	1.24088	+	TCPopen OK(11) TCPclose
23	197	1.79745	+	TCPopen OK(9) TCPclose
24	342	3.12044	+	TCPopen OK(7) TCPclose
25	631	5.7573	+	TCPopen OK(5) TCPclose
26	1199	10.9398	-	TCPopen TCPclose
27	8075	73.677	-	TCPopen OK TCPreset
27 Start with: TCPopen				
21 End with: TCPclose (2,768 samples)				
2 End with: TCPreset (8,090 samples)				
3 End with: TCPtimeout (8 samples)				
1 End with: NIDSexit (94 samples)				
18 Positive sample Types. 1,508 Positive samples.				
9 Negative sample Types. 9,452 Negative samples.				
10,960 Total Samples				
27 Unique Types				

Data Summary: POP3 Composite Alphabet total.pcap				
Type	Count	Percent	Label	Sample
1	1	0.00912409	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK RETR OK QUIT OK TCPclose
2	1	0.00912409	-	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPtimeout
3	1	0.00912409	-	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE RETR DELE QUIT TCPtimeout
4	1	0.00912409	-	TCPopen QUIT OK(2) TCPclose
5	1	0.00912409	-	TCPopen QUIT OK(2) TCPtimeout
Continued on next page				

Type	Count	Percent	Label	Sample
6	1	0.00912409	-	TCPopen QUIT TCPclose
7	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
8	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
9	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
10	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
11	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
12	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
13	2	0.0182482	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK QUIT OK TCPclose
14	2	0.0182482	-	TCPopen OK USER TCPreset
15	5	0.0456204	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
16	5	0.0456204	-	TCPopen TCPtimeout
17	9	0.0821168	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK TCPclose
18	15	0.136861	-	TCPopen TCPreset

Type	Count	Percent	Label	Sample
19	25	0.228102	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
20	32	0.291971	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
21	47	0.428832	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
22	60	0.547445	-	TCPopen OK(2) ERR TCPclose
23	69	0.629562	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
24	94	0.857664	-	TCPopen NIDSexit
25	136	1.24088	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
26	197	1.79745	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK RETR OK DELE OK QUIT OK TCPclose
27	342	3.12044	+	TCPopen OK USER OK PASS OK STAT OK RETR OK DELE OK QUIT OK TCPclose
28	631	5.7573	+	TCPopen OK USER OK PASS OK STAT OK QUIT OK TCPclose
29	1198	10.9307	-	TCPopen TCPclose
30	8073	73.6588	-	TCPopen OK TCPreset
30 Start with: TCPopen				
22 End with: TCPclose (2,768 samples)				
3 End with: TCPreset (8,090 samples)				
4 End with: TCPtimeout (8 samples)				
1 End with: NIDSexit (94 samples)				
18 Positive sample Types. 1,508 Positive samples.				
12 Negative sample Types. 9,452 Negative samples.				
10,960 Total Samples				
30 Unique Types				

Appendix B. Low Level Implementation

This appendix introduces supporting toolkits we reviewed, as well as the environment and tools used in the development process. Finally, we briefly introduce a low level look at flowtool and flowinfer.

B.1 Sources for Protocol Formats

There are rich repositories we can mine for protocol format information. Several open source projects for trace collection and intrusion detection provide source code.

Wireshark contains a large body of protocol formats (approximately 700 at this time) that have been gleaned from open specifications, source code, and community reverse engineering efforts [48]. JNetStream specifies network protocol formats in a description language called Network Protocol Language (NPL) [19]. The NPL specifications are compiled to Java class files by the included nplc compiler [19]. Netdude supports protocol formats that are hardcoded in c source and PHDL, a packet header description language [133]. Both Wireshark and jNetStream are protocol analyzers while Netdude concentrates on packet trace manipulation and presentation over analysis.

Bro is a research oriented Intrusion Detection System [140]. Bro provides the protocol description language binpac, a binpac language compiler, and protocol descriptions with its source code [140, 192]. The binpac protocol descriptions cover a range of ASCII text and binary protocols. Snort is another popular open source IDS that embodies hand coded application layer protocol parsers [98].

There are also several data description languages that are potentially useful for describing ad hoc mixed binary and text data such as protocol formats. Fisher et al propose automated inference of ad hoc data to generate PADS data description language [81]. PacketTypes is another data description language specifically designed for specifying network protocol messages [164]. DataScript is a specification language for binary data formats [13]. Other options include direct use of Augmented BNF or Abstract Syntax Notation number One (ASN.1).

B.2 Automata Toolkits

There are several frameworks that contain foundational language theoretic elements but do not include GI algorithms.

B.2.1 AMoRE. AMoRE [161] is used by Berg [23, Section 4] as the basis for both a direct implementation and domain optimized version of Angluin’s learner. The AMoRE library is implemented as a library in portable C. While the AMoRE source code is available the build system was not compatible with the build system on our development system. For testing purposes we ported the build system to autoconf¹. AMoRE was also used by the MERLin Project [267].

B.2.2 Vaucanson. Vaucanson² is a C++ based automata library maintained by the EPITA Research and Development Laboratory (the same research group also maintains the Mical GI package) [86]. The version available to the public at the time of writing is 1.1.1. Lombardy [152] explains Vaucanson in detail and provides comparison to the AMoRE library. An early version of Vaucanson provides the automata engine for the Mical GI package [213].

B.2.3 JFLAP. JFLAP is an automata learning tool implemented in Java that supports many of the core operations required for grammatical inference [218, 219]. Source code for version 4 is available for download but source code for newer versions 5 and 6 must be requested from the author [219]. JFLAP 4 is used in the kBehavior algorithm implementation [158, 198].

B.2.4 Grail. Grail [276] and it’s more recent incarnation as Grail+ [285] provide a range of useful automata operations. Unfortunately, the C++ source code is rather dated and the build system does not support our development platform. A modified version of Grail version 2.5 was created by the MERLin Project to study

¹Autoconf – <http://www.gnu.org/software/autoconf/>

²Vaucanson < Project – <http://www.lrde.epita.fr/cgi-bin/twiki/view/Projects/Vaucanson>

generalized NFA [267]. The modified version provides build targets for Linux but not for contemporary versions of GCC.

B.3 Grammatical Inference Implementations

While there is no single definitive collection of GI algorithms source code for several of the algorithms discussed above are available. Also, LearnLib and Mical frameworks each provide a subset of the GI algorithms.

B.3.1 LearnLib. LearnLib³ is a C++ library [211] that provides an architecture for GI algorithm testing. Because the library is designed using *Common Object Request Broker Architecture* (CORBA)⁴ interfaces it can be used from any language with CORBA support. At this time it only implements Angluin's L^* and a modified L^* . LearnLib is incorporated into the *Formal Methods for Industrial Critical Systems-Java Electronic Tool Integration* (FMICS-jETI⁵) platform to support model design recovery with Smyle [157]. While the CORBA interface is currently exposed on the Internet and sample uses are available in example source code; the source code for the library does not appear to be available to the public.

B.3.2 Mical. Mical is a C++ library of grammatical inference algorithms including k -TSSI, k -RI, MGCI, and RPNI [213]. The library extensively utilizes C++ template mechanisms that are compatible with GCC⁶ version 3.2 and version 3.3 but not newer versions of the GCC toolchain (currently version 4.2.2). The only publicly released version of the software is version 0.1 from 10 July 2003 [213]. The library also implements a range of functions internally including: creation of MCA, creation of PTA, and conversion of MCA to PTA [213]. We attempted, unsuccessfully, to build the library with relaxed C++ dialect options ⁷ available in GCC version 4.2.2.

³LearnLib – <http://faelis.cs.uni-dortmund.de/>

⁴CORBA FAQ – <http://www.omg.org/gettingstarted/corbafaq.htm>

⁵FMICS-jETI – <http://jeti.cs.uni-dortmund.de/fmics/>

⁶GCC, the GNU Compiler Collection – <http://gcc.gnu.org>

⁷-fpermissive, etc..., see [85, Section 3.5]

We were able to build and evaluate the library by creating a parallel installation of GCC version 3.3.6. Mical uses a modified version of the Vaucanson automata library included with its source code and is not compatible with current versions of Vaucanson [86].

B.3.3 Other Implementations. Implementations of several other algorithms publicly available. Ammons provides C source code for k -tails and sk-strings for PFSA inference [6]. C source code for Algeria, a stochastic automata inference algorithm [94], RPNI, and k -tails is available from [225]. Mariani and Pezzé provide source for kBehavior and kTails⁸. Their algorithms are implemented in Java [158, 198] and use JFLAP version 4 for automata support [219].

B.4 Implementation Language

The choice of implementation language was driven by the selection of support packages and our familiarity with C and C++. The choice of development environment was driven by the implementation language. We primarily used Eclipse⁹ version 3.2, KDevelop¹⁰ version 3.5.0, and various text editors on an openSUSE¹¹ version 10.3 Linux computer. We used Concurrent Versions System (CVS) for revision control.

B.5 Low Level Implementation

Our proof of concept low level implementation is comprised for two programs, flowtool and flowinfer, and associated driver scripts. Flowtool process raw pcap trace files and produces alphabet and sample string files for input into flowinfer.

B.5.1 flowtool. Flowtool is a C language program using GLib version 2.14¹² standard data structures for dynamic storage and libnids for TCP connection re-

⁸k-tails is also covered by [99]

⁹Eclipse – <http://www.eclipse.org>

¹⁰KDevelop – <http://www.kdevelop.org/>

¹¹openSUSE – <http://www.opensuse.org>

¹²GLib Reference Manual - <http://library.gnome.org/devel/glib/2.14/>

Figure B.1: Flowtool verbose sample output.

```
# Sample Type-6
# Protocol: 25 Server 172.16.114.50 Client 152.204.242.193.1941
# Key = 172.16.114.50:25,152.204.242.193:1941: 923693227.228408
# Start: 923693227.228408
# End: 923693238.320118
# Wireshark filter:(ip.addr eq 172.16.114.50 and ip.addr eq 152.204.242.193) and (tcp.port eq 25 and tcp.port eq 1941)
- TCPopen 220 250 250 503 HELP 500 221 TCPclose

# Sample Type-32
# Protocol: 25 Server 172.16.114.50 Client 202.49.244.10.1027
# Key = 172.16.114.50:25,202.49.244.10:1027: 922715292.597701
# Start: 922715292.597701
# End: 922715294.666466
# Wireshark filter:(ip.addr eq 172.16.114.50 and ip.addr eq 202.49.244.10) and (tcp.port eq 25 and tcp.port eq 1027)
- TCPopen 220 250 250 503 HELP 500 221 TCPclose

# Sample Type-32
# Protocol: 25 Server 172.16.114.50 Client 202.49.244.10.1027
# Key = 172.16.114.50:25,202.49.244.10:1027: 922715290.284924
# Start: 922715290.284924
# End: 922715292.352719
# Wireshark filter:(ip.addr eq 172.16.114.50 and ip.addr eq 202.49.244.10) and (tcp.port eq 25 and tcp.port eq 1027)
- TCPopen 220 250 250 503 HELP 500 221 TCPclose

.
```

assembly and defragmentation. An elided call graph from main of flowtool is shown in Figure B.2.

As the libnids library creates the connection level flow the application session samples are incrementally created by parsing the application data flow. As previously mentioned, because the code is a proof of concept we chose to implement hand-coded operator parsers derived from specification documents and heuristics from Wireshark and Bro. Once the application level traffic is parsed we write out the alphabet of the protocol under consideration and the sample strings of operators, replies, and the composite of operators intermixed with replies. The different sample types are written to separate flat text files. Flowtool verbose sample output is shown in Figure B.1

Flowtool was developed using the KDevelop IDE and autoconf build system. The C source code supports the Doxygen documentation system.

B.5.2 flowinfer. We developed the C++ language program called flowinfer for automata inference. We chose the mical GI toolkit as the basis of flowinfer. Our choice of mical was driven by the fact that it was the only GI toolkit that supported multi-letter alphabets. Additionally, mical was the only toolkit with an integrated automata toolkit. The program processes the output flat text files from flowinfer using

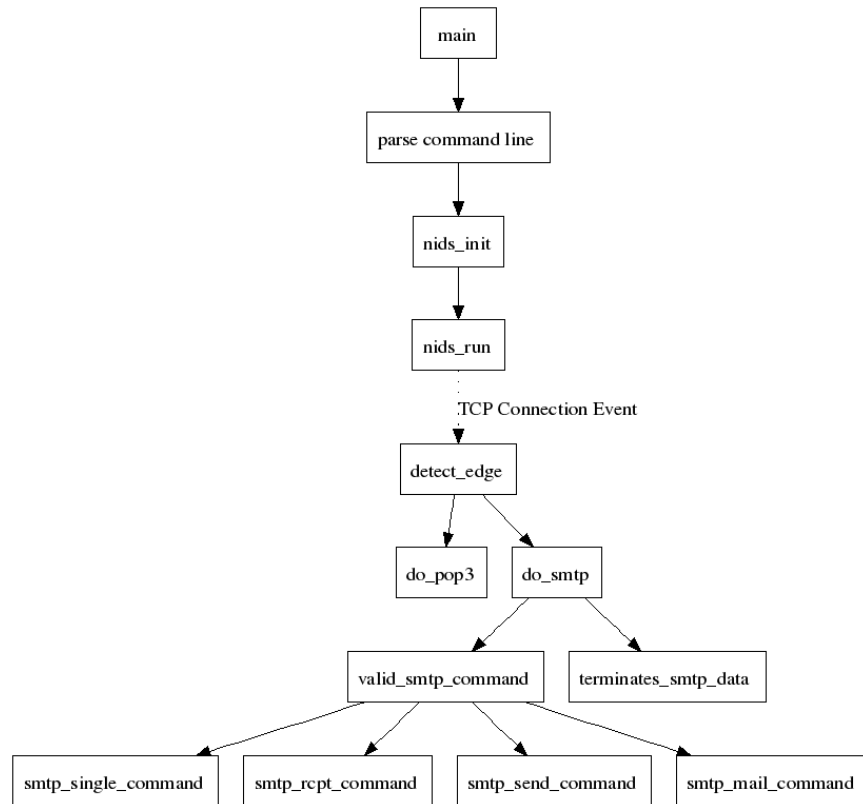


Figure B.2: Flowtool call graph (elided) - libnids creates a packet processing event loop that dispatches events to registered callback functions. The callback function named `detect_edge` is registered for TCP connection events. The `detect_edge` callback determines if the connection is on a port of interest and calls the appropriate protocol data format parser in either `do_smtp` or `do_pop3`.

micals k -RI and k -TSSI algorithm implementations to generate Vaucanson automata. The automata are output as GraphViz¹³ dot files and post-processed to graphics by a driver script called *flowtool_data*. The driver script builds the command line parameters and invokes flowinfer for the selected algorithms for k values 1 to 5. A summary of the automata states, edges, initial states, and terminal/final states is generated from the output of flowinfer for analysis purposes. Low-level data structures in mical and the integrated version of Vaucanson are implemented using the C++ standard template library.

Flowinfer was developed using the KDevelop IDE and autoconf build system. The C++ source code supports the Doxygen documentation system.

¹³GraphViz - <http://www.graphviz.org>

Appendix C. Inferred Automaton

This appendix presents the automaton inferred by the k -RI and k -TSSI algorithms for the subset of the POP3 protocols exercised by the IDEVAL data set. Inferred SMTP automaton are not presented here because the graphs were not legible if we scaled the figures to fit on a single page.

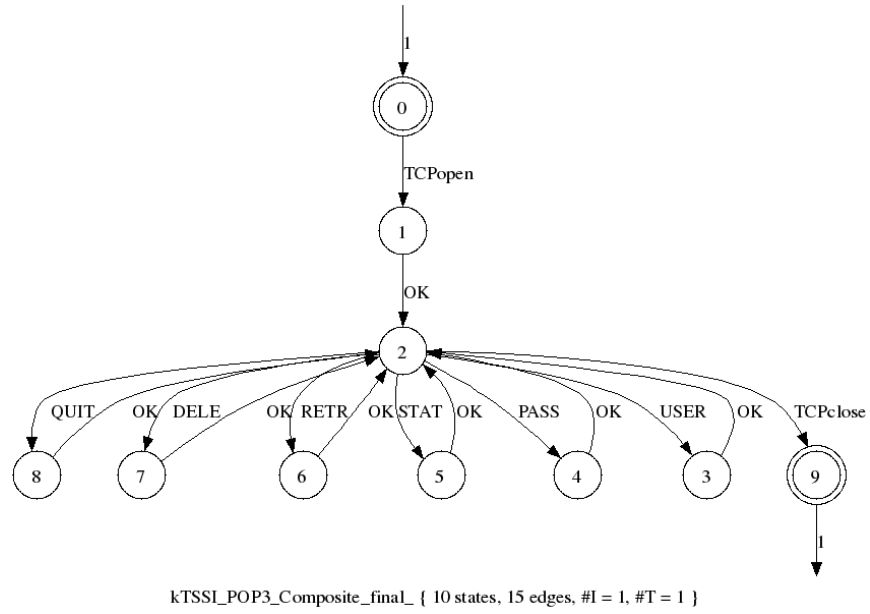


Figure C.1: k -TSSI POP3 Composite Final Automaton $k = 1$. k -TSSI with $k = 1$ over-restricts the target automaton. Session initiation and session termination are not inferred as separate states.

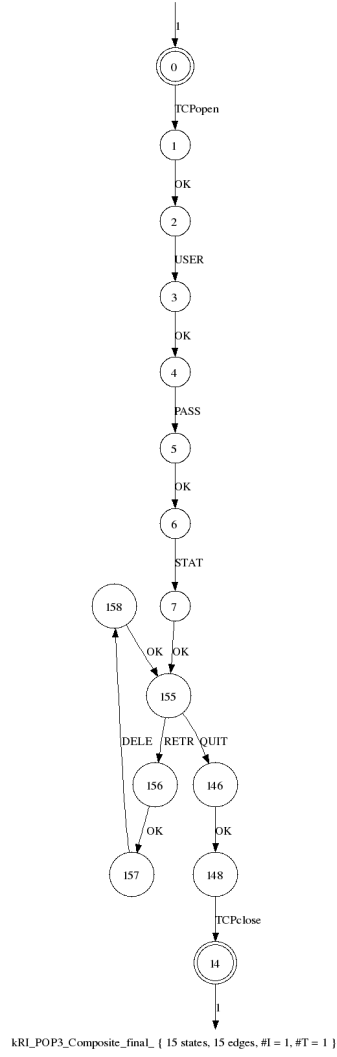


Figure C.2: k -RI POP3 Composite Final Automaton $k = 1$. k -RI inference exactly matches the target automaton for the subset of POP3 specification exercised by the IDEVAL data set.

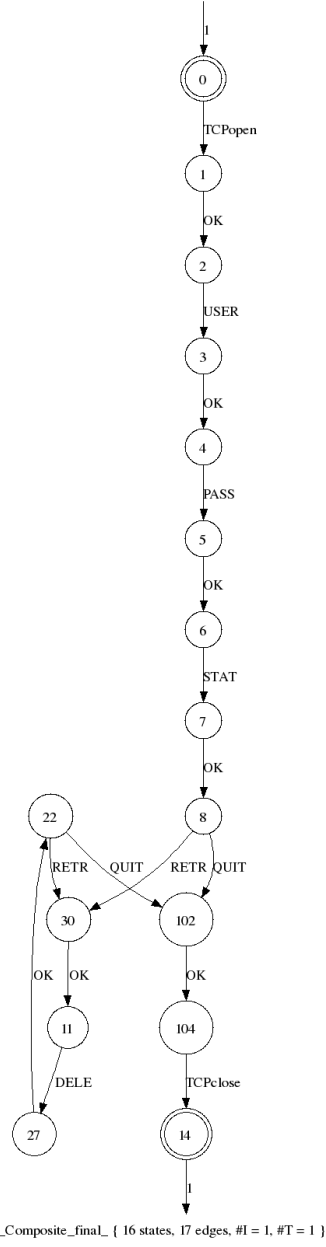


Figure C.3: k -RI POP3 Composite Final Automaton $k = 2$.
 k -RI inference over-generalizes the target automaton.

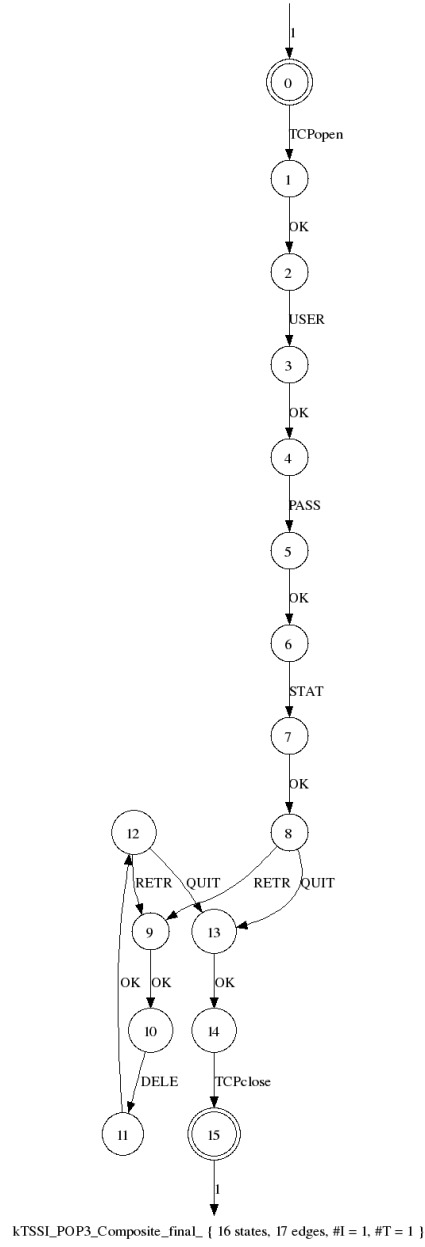


Figure C.4: k -TSSI POP3 Composite Final Automaton $k = 2$. k -TSSI with $k = 2$ exactly matches the k -RI inference with $k = 2$. The hypothesis automaton over-generalizes the target automaton.

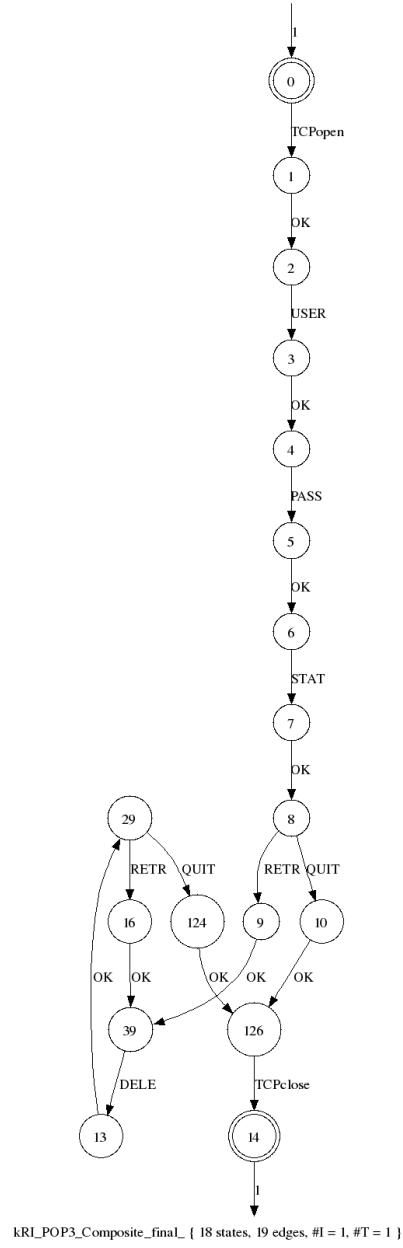


Figure C.5: k -RI POP3 Composite Final Automaton $k = 3$. k -RI inference continues to over-generalizes the target automaton.

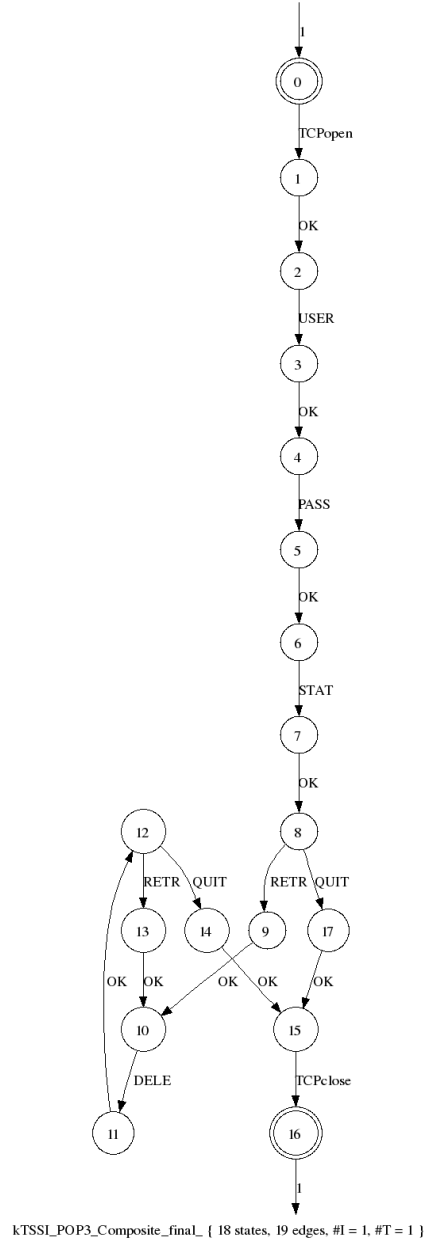


Figure C.6: k -TSSI POP3 Composite Final Automaton $k = 3$. k -TSSI with $k = 3$ exactly matches the k -RI inference with $k = 3$. It also over-generalizes the target automaton.

Bibliography

1. Abbes, T. and M. Rusinowitch. “Fast Multipattern Matching for Intrusion Detection”. Urs E. Gattiker (editor), *EICAR 2004 Conference CD-rom: Best Paper Proceedings*. 2004. ISBN 8798727168.
2. Ahn, Gordon. “MSN Protocol Analyzer”, February 2005. URL <http://www.securityfocus.com/tools/3814>.
3. Aho, Alfred V. and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. 1: Parsing*. Prentice-Hall, Englewood Cliffs, 1972. This book is available for download through the ACM Portal at <http://portal.acm.org/citation.cfm?id=SERIES11430.578789>.
4. Aizenstein, Howard, Tibor Hegedűs, Lisa Hellerstein, and Leonard Pitt. “Complexity theoretic hardness results for query learning”. *Comput. Complex.*, 7(1):19–53, 1998. ISSN 1016-3328.
5. Alur, Rajeev, Kousha Etessami, and Mihalis Yannakakis. “Realizability and verification of MSC graphs”. *Theoretical Computer Science*, 331(1):97–114, 15 February 2005.
6. Ammons, Glenn, Rastislav Bodík, and James R. Larus. “Mining specifications”. *POPL ’02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 4–16. ACM Press, New York, NY, USA, 2002. ISBN 1581134509.
7. Angluin, Dana. “On the complexity of minimum inference of regular sets”. *Information and Control*, 39(3):337–350, 1978.
8. Angluin, Dana. “Finding patterns common to a set of strings (Extended Abstract)”. 130–141, 1979.
9. Angluin, Dana. “Inference of Reversible Languages”. *Journal of the ACM*, 29(3):741–765, 1982.
10. Angluin, Dana. “Learning regular sets from queries and counterexamples”. *Inf. Comput.*, 75(2):87–106, 1987. ISSN 0890-5401.
11. Angluin, Dana. “Queries and Concept Learning”. *Machine Learning*, 2(4):319–342, 4 January 1988.
12. Aycock, John Daniel. *Computer viruses and malware*, volume 22 of *Advances in information security*. Springer, New York, 2006. ISBN 9780387302362.
13. Back, Godmar. “DataScript - A Specification and Scripting Language for Binary Data”. *GPCE ’02: Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering*, 66–77. Springer-Verlag, London, UK, 2002. ISBN 3540442847. URL <http://people.cs.vt.edu/~gback/>.

14. Balcázar, José L., Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. “A Note on the Query Complexity of Learning DFA (Extended Abstract)”. *ALT '92: Proceedings of the Third Workshop on Algorithmic Learning Theory*, 53–62. Springer-Verlag, London, UK, 1993. ISBN 3540573690.
15. Barford, Paul and David Plonka. “Characteristics of network traffic flow anomalies”. *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 69–73. ACM, New York, NY, USA, 2001. ISBN 1581134355.
16. Barnard, J., J. Whitworth, and M. Woodward. “Communicating X-machines”. *Information and Software Technology*, 38(6):401–407, June 1996.
17. Barnard, Judith. “COMX: A Methodology for the formal design of computer systems using Communicating X-machines”. URL <http://www.soc.staffs.ac.uk/research/groups/comx/>.
18. Beddoe, Marshall. “Protocol Informatics”, 2004. A mirror of the project web site is available at: <http://www.4tphi.net/~awalters/PI/PI.html>.
19. Bednarczyk, Mark. “jNetStream — OpenSource Protocol Analyzer and Decoder SDK”, 2007. URL <http://jnetstream.sourceforge.net/>.
20. Belenky, A. and N. Ansari. “Accommodating fragmentation in deterministic packet marking for IP traceback”. *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*. 2003.
21. Belz, Anja and Berkan Eskikaya. “A Genetic Algorithm for Finite State Automata Induction with an Application to Phonotactics”. *Proceedings of the ESSLLI-98 Workshop on Automated Aquisition of Syntax and Parsing*. 1998.
22. Berg, Therese, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. “On the Correspondence Between Conformance Testing and Regular Inference”. *8th International Conference, FASE 2005, Held as Part of the Joint Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005.*, Lecture Notes in Computer Science, 175–189. Springer Berlin / Heidelberg, 2005. ISBN 9783540254201.
23. Berg, Therese, Bengt Jonsson, Martin Leucker, and Mayank Saksena. *Insights to Angluin’s Learning*. Technical Report 2003-039, Uppsala Universitet, 2003. URL <http://www.it.uu.se/research/publications/reports/2003-039/>.
24. Bhargavan, K., S. Chandra, P. J. McCann, and C. A. Gunter. “What packets may come: automata for network monitoring”. *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 206–219, 2001.
25. Bhargavan, Karthikeyan. “Network Event Recognition”, 2003.
26. Bhargavan, Karthikeyan and Carl A. Gunter. “Network Event Recognition for Packet-Mode Surveillance”, 2002.

27. Bhargavan, Karthikeyan and Carl A. Gunter. “Requirements for a Practical Network Event Recognition Language”, 2002.
28. Biermann, A. and J. Feldman. “On the synthesis of finite state machines from samples of their behavior”. *IEEE Transactions on Computers*, 21(6):592–597, 1972.
29. Biondi, Philippe. “Scapy”, 2007. URL <http://www.secdev.org/projects/scapy/>.
30. Bishop, Steve, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. “Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations”. *SIGPLAN Not.*, 41(1):55–66, 2006. ISSN 0362-1340.
31. Bollig, Benedikt, Joost-Pieter Katoen, Carsten Kern, and Martin Leucker. *Re-playing Play in and Play out: Synthesis of Design Models from Scenarios by Learning*. Technical Report AIB-2006-12, RWTH Aachen, Germany, October 2006. URL <http://smyle.in.tum.de/report.html>.
32. Borison, Nikita, David J. Brumley, Helen J. Wang, John Dunagan, Pallavi Joshi, and Chuanxiong Guo. *A Generic Application-Level Protocol Analyzer and its Language*. Technical report, Microsoft Research, 2005. United States Patent 20070112969.
33. Bosworth, Edward. “Moore and Mealy Machines”, 2007. URL <http://csc.colstate.edu/bosworth/>.
34. Brendan Gregg, Michael B. “Chaosreader”, 02 May 2004. URL <http://www.brendangregg.com/chaosreader.html>.
35. Brownlee, N. and M. Murray. “Streams, flows and torrents”, 2001. URL <http://www.caida.org/publications/papers/2001/StreamsFlowsTorrents/>.
36. Bunke, Horst. and Alberto. Sanfeliu. *Syntactic and structural pattern recognition: theory and applications*, volume 7 of *Series in computer science*. World Scientific, Singapore, 1990. ISBN 9971505665.
37. Caron, Pascal. “LANGAGE: A Maple package for automation characterization of regular languages”. *Theoretical Computer Science*, 231(1):5–15, 2000. ISSN 0304-3975.
38. Chapman, Matthew. “rdesktop: A Remote Desktop Protocol client”, 13 September 2006. URL <http://sourceforge.net/projects/rdesktop/>.
39. Chen, Xiao Jun and Hasan Ural. “Construction of Deadlock-free Designs of Communication Protocols from Observations”. *The Computer Journal*, 45(2):162, 2002.
40. Chikofsky, Elliot J. and James H. Cross II. “Reverse Engineering and Design Recovery: A Taxonomy”. *IEEE Software*, 7(1):13–17, 1990.

41. Cicchello, Orlando and Stefan C. Kremer. “Inducing grammars from sparse data sets: a survey of algorithms and results”. *J. Mach. Learn. Res.*, 4:603–632, 2003. ISSN 1533-7928.
42. Claffy, K. C, H.-W. Braun, and G. Polyzos. *A parameterizable methodology for Internet traffic flow profiling*. Technical report, Applied Network Research Center, San Diego Supercomputer Center, San Diego, CA 92186-9784, 1994. URL <http://www.caida.org/publications/papers/1994/itf/>.
43. Clark, Alexander. “GISK project page: Grammatical Infernece with String Kernels”, November 2006. URL <http://www.cs.rhul.ac.uk/home/alexc/gisk/>.
44. Clark, Alexander. “PAC-learning unambiguous NTS languages”. *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*, 59–71. 2006. URL <http://www.cs.rhul.ac.uk/home/alexc/clark.html>.
45. Clark, Alexander, Christophe Costa Florêncio, and Chris Watkins. “Languages as Hyperplanes: grammatical inference with string kernels”. *Proceedings of the European Conference on Machine Learning (ECML)*, 90–101. Springer, 2006. URL <http://www.cs.rhul.ac.uk/home/alexc/clark.html>.
46. Clark, Alexander, Christophe Costa Florêncio, Chris Watkins, and Mariette Serayet. “Planar Languages and Learnability”. *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*, 148–160. Springer, 2006. URL <http://www.cs.rhul.ac.uk/home/alexc/clark.html>.
47. Clarke, Gordon R., Deon. Reynders, and Edwin BSc Wright. *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Practical professional books from Elsevier. Newnes, Oxford, 2004. ISBN 0750657995.
48. Combs, Gerald. “About Wireshark”, August 2007. URL <http://www.wireshark.org/about.html>.
49. Comer, Douglas. *Internetworking with TCP/IP, Vol 1 (5th Edition)*. Prentice Hall, 2005. ISBN 0131876716.
50. Contributors, Pidgin. “About — Pidgin”, 2007. URL <http://www.pidgin.im/about/>.
51. Contributors, TCPDUMP. “TCPDUMP public repository”, September 2006. URL <http://www.tcpdump.org/>.
52. Cook, Carl L. R. “rdp vs w2k”, November 2000. URL <http://www.rdesktop.org/archive/2000/msg00038.html>.
53. Cook, Jonathan E. and Alexander L. Wolf. “Discovering models of software processes from event-based data”. *ACM Trans.Softw.Eng.Methodol.*, 7(3):215–249, 1998.
54. Coste, Franç, Daniel Fredouille, Christopher Kermorvant, and Colin de la Higuera. “Introducing Domain and Typing Bias in Automata Inference”, 2004. URL <http://www.irisa.fr>.

55. Coste, François and Daniel Fredouille. *What is the Search Space for the Inference of Non-Deterministic, Unambiguous and Deterministic Automata?* Technical Report RR-4907, N°: IRISA/INRIA, 2003. URL <http://hal.inria.fr/inria-00071673/en/>.
56. Cottrell, Les. “Network Monitoring Tools”, August 2007. URL <http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html>.
57. Croker, De. and P. Overell. “Augmented BNF for Syntax Specifications: ABNF”, October 2005. URL <http://tools.ietf.org/html/rfc4234>.
58. Cui, Weidong, Marcus Peinado, and Helen J. Wang. “Discoverer: Automatic Protocol Reverse Engineering from Network Traces”, August 2007. URL <http://research.microsoft.com/~wdcui/>.
59. Dallmeier, Valentin, Christian Lindig, Andrzej Wasylkowski, and Andreas Zeller. “Mining object behavior with ADABU”. *WODA '06: Proceedings of the 2006 international workshop on Dynamic systems analysis*, 17–24. ACM Press, New York, NY, USA, 2006. ISBN 1595934006.
60. Danyi, G. “Regular inference with maximal valid grammar method”. *Grammatical Inference: Theory, Applications and Alternatives, IEE Colloquium on*, 5/1–5/9. 22–23 April 1993.
61. de la Higuera, Colin. “A bibliographic study of grammatical inference”. *Pattern Recognition*, 38:1332–1348, September 2005. URL <http://labh-curien.univ-st-etienne.fr/~cdlh/>.
62. de la Higuera, Colin. “Ten open problems in grammatical inference”. *Proceedings of ICGI 2006, Tokyo LNCS 4201 32–44*. 2006. URL <http://labh-curien.univ-st-etienne.fr/~cdlh/webpages/publications.html>.
63. Deraleau, Jason. “Inside Samba: Windows Sharing for the Mac”, 18 March 2003. URL <http://www.macdevcenter.com/lpt/a/3323>.
64. Deri, Luca. “What is ntop?”, 20 July 2007. URL <http://www.ntop.org/overview.html>.
65. Diestel, Reinhard. *Graph theory*. New York : Springer, 3rd edition, 2006. ISBN 0387989765.
66. Dongxi, Liu, Li Xiaoyong, and Bai Yingcai. “An attack-finding algorithm for security protocols”. *J. Comput. Sci. Technol.*, 17(4):450–463, 2002. ISSN 1000-9000.
67. Drechsler, Rolf. *Advanced formal verification*. Kluwer Academic Publishers, Boston, 2004. ISBN 1402077211.
68. Dssouli, R., K. Saleh, E. Aboulhamid, A. En-Nouaary, and C. Bourhfir. “Test development for communication protocols: towards automation”. *Comput. Networks*, 31(17):1835–1872, 1999. ISSN 1389-1286.

69. Dupont, Pierre. “Incremental regular inference”. *ICG! ’96: Proceedings of the 3rd International Colloquium on Grammatical Inference*, 222–237. Springer-Verlag, London, UK, 1996.
70. Dupont, Pierre, Laurent Miclet, and Enrique Vidal. “What Is the Search Space of the Regular Inference?” *ICGI ’94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, 25–37. Springer-Verlag, London, UK, 1994. ISBN 3540584730.
71. Earley, Jay. “An efficient context-free parsing algorithm”. *Communications of the ACM*, 13(2):94–102, 1970. ISSN 0001-0782.
72. Eilam, Eldad. *Reversing: Secrets of Reverse Engineering*. Wiley, 2005. ISBN 0764574817.
73. Elson, Jeremy. “tcpflow – TCP Flow Recorder”, August 2003. URL <http://www.circlemud.org/~jelson/software/tcpflow/>.
74. Emerald, J. D., K. G. Subramanian, and D. G. Thomas. “Learning code regular and code linear languages”. *ICG! ’96: Proceedings of the 3rd International Colloquium on Grammatical Inference*, 211–221. Springer-Verlag, London, UK, 1996. ISBN 3540617787.
75. Erickson, Jon. *Hacking: The Art of Exploitation*. No Starch, 2003. ISBN 1593270070.
76. Erman, Jeffrey, Anirban Mahanti, and Martin Arlitt. “Byte me: a case for byte accuracy in traffic classification”. *MineNet ’07: Proceedings of the 3rd annual ACM workshop on Mining network data*, 35–38. ACM, New York, NY, USA, 2007. ISBN 9781595937926.
77. EventHelix.com Inc. “EventHelix.com - Sequence Diagram Based System Design Tool”, 2007. URL <http://www.eventhelix.com/>.
78. Farmer, Mick. “Transport Control Protocol (TCP)”, 24 November 2005. URL <http://penguin.dcs.bbk.ac.uk/academic/networks/>.
79. Fernandez, John D. and Andres E. Fernandez. “SCADA systems: vulnerabilities and remediation”. *J. Comput. Small Coll.*, 20(4):160–168, 2005. ISSN 1937-4771. URL <http://portal.acm.org/citation.cfm?id=1047846.1047872#>.
80. Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. “Hypertext Transfer Protocol – HTTP/1.1”, June 1999. URL <http://tools.ietf.org/html/rfc2616>.
81. Fisher, Kathleen, David Walker, Kenny Q. Zhu, and Peter White. “From Dirt to Shovels: Fully Automatic Tool Generation from Ad Hoc Data”. *POPL*, January 2008. URL <http://www.padsproj.org>.
82. Forsberg, Erik. “Reverse Engineering and Implementation of the RDP 5 Protocol”, 5 March 2004. URL <http://efod.se/writings/thesis.pdf/view>.

83. Foster, James C. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Syngress Publishing, 2007. ISBN 1597490741.
84. Foster, Vincent T. Liu James C. *Writing Security Tools and Exploits*. Syngress Publishing, 2007.
85. Foundation, Free Software. “Using the GNU Compiler Collection (GCC)”. URL <http://gcc.gnu.org/onlinedocs/gcc-4.2.2/gcc/>.
86. Fouquier, Geoffroy. “The Vaucanson project”, 24 January 2006. URL <http://www.lrde.epita.fr/cgi-bin/twiki/view/Projects/Vaucanson>.
87. Fravia. “Fravia’s web-searching lore”, 2006. URL <http://www.searchlores.org/basic.htm>.
88. Freund, Yoav, Michael Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. “Efficient learning of typical finite automata from random walks”. *STOC ’93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 315–324. ACM, New York, NY, USA, 1993. ISBN 0897915917.
89. Garcia, P. and E. Vidal. “Inference of k-testable languages in the strict sense and application to syntactic pattern recognition”. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, volume 12, 920–925. September 1990.
90. Garcia, Pedro, Enrique Vidal, and Francisco Casacuberta. “Local languages, the successor method, and a step towards a general methodology for the inference of regular grammars”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):841–845, 1987. ISSN 0162-8828.
91. Gebski, M., A. Penev, and R. K. Wong. *Data Warehousing and Knowledge Discovery*, chapter Classification of hidden network streams, 332–341. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2006. AN: 9144529.
92. Giordano, Jean-Yves. “Grammatical inference using tabu search”. *ICG! ’96: Proceedings of the 3rd International Colloquium on Grammatical Inference*, 292–300. Springer-Verlag, London, UK, 1996. ISBN 3540617787.
93. Glaser, M. Gerhard. “Schulung ” Heterogene Netze mit TCP/IP”, March 2003.
94. Goan, T., N. Benson, and O. Etzioni. “A grammar inference algorithm for the World Wide Web”, 1996.
95. Gold, Mark E. “Language identification in the limit”. *Information and Control*, 10:447, 1967.
96. Gouda, M. G., E. M. Gurari, T. H. Lai, and L. E. Rosier. “On deadlock detection in systems of communicating finite state machines”. *Comput. Artif. Intell.*, 6(3):209–228, 1987. ISSN 0232-0274.

97. Graine, S. "Supervised learning of regular languages by neural networks". *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 2. 1994. ISBN 078031901X.
98. Gregg, Michael B. and Stephen Watkins. *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network*. Syngress Publishing, 2006. ISBN 1597491098.
99. Gronfors, T. and M. Juhola. "Experiments and comparison of inference methods of regular grammars". *Systems, Man and Cybernetics, IEEE Transactions on*, volume 22, 821–826. July 1992.
100. Guha, B. and B. Mukherjee. "Network security via reverse engineering of TCP code: vulnerability analysis and proposed solutions". *IEEE Network*, 11(4):40, July 1997.
101. Hagerer, Andreas, Hardi Hungar, Oliver Niese, and Bernhard Steffen. "Model Generation by Moderated Regular Extrapolation". *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, 80–95. Springer-Verlag, London, UK, 2002. ISBN 3540433538.
102. Haussler, David. "Part 1: Overview of the Probably Approximately Correct (PAC) Learning Framework".
103. Heilbrun, Brad. "Re: 128bit Encryption Compatibility", 18 April 2002.
104. Helin, A., J. J. Viide, and M. Laakso. "PROTOS Protocol Genome Project", August 2006. URL <http://www.ee.oulu.fi/research/ouspg/protos/genome/>.
105. Hertel, Christopher. *Implementing CIFS: the common Internet file system*. Upper Saddle River, N.J. ; Prentice Hall PTR, c2004., 2004. ISBN 013047116X. URL <http://ubiqx.org/cifs/>.
106. Hingston, P. "Inference of regular languages using model simplicity". *Computer Science Conference, 2001. ACSC 2001. Proceedings. 24th Australasian*, 69–76. 29 January–4 February 2001.
107. Hoffmann, Petr. "Improving RPNI algorithm using minimal message length". *AIAP'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference*, 378–383. ACTA Press, Anaheim, CA, USA, 2007.
108. Hoglund, Greg and Gary McGraw. *Exploiting software: how to break code*. Boston : Addison-Wesley, c2004., 2004. ISBN 0201786958.
109. Holzmann, Gerard J. *Design and validation of computer protocols*. Englewood Cliffs, N.J. : Prentice Hall, c1991., 1991. ISBN 0135399254. URL <http://spinroot.com/spin/Doc/Book91.html>.
110. Hopcroft, John E., Rajeev. Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Boston, 2nd edition, 2001. ISBN 0201441241.

111. IEEE USA Board of Directors. “IEEE USA Position Statement on Reverse Engineering”, June 2003. URL <http://ieeeusa.com/policy/positions/reverse.html>.
112. IETF Network Working Group. “Post Office Protocol - Version 3”, May 1996. URL <http://tools.ietf.org/html/rfc1939>.
113. IETF Network Working Group. “Simple Mail Transfer Protocol”, April 2001. URL <http://tools.ietf.org/html/rfc2821>.
114. Ingham, Kenneth L., Anil Somayaji, John Burge, and Stephanie Forrest. “Learning DFA representations of HTTP for protecting web applications”. *Computer Networks*, 51(5):1239–1255, 2007.
115. Ingham, Kenneth LeRoy III. *Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy*. Doctor of philosophy computer science, The University of New Mexico, Albuquerque, New Mexico, May 2007. URL <http://hdl.handle.net/1928/2874>.
116. Internet Assigned Numbers Authority. “Port Numbers”, 03 December 2007. URL <http://www.iana.org/assignments/port-numbers>.
117. ITU-T. “Z.120 Message Sequence Charts (MSC)”, March 2004. URL <http://www.itu.int/ITU-T/studygroups/com17/languages/>.
118. Jain, R. and S. Routhier. “Packet Trains—Measurements and a New Model for Computer Network Traffic”. *Selected Areas in Communications, IEEE Journal on*, 4(6):986–995, 1986.
119. Jaiswal, S., G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. “Inferring TCP connection characteristics through passive measurements”, 2004.
120. Javed, Faizan. “Inferring context-free grammars for domain-specific languages”. *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 212–213. ACM, New York, NY, USA, 2005. ISBN 1595931937.
121. Jourdan, G. V., H. Ural, and H. Yenigun. “Recovering the lattice of repetitive sub-functions”. *Lecture notes in computer science*, 956–965, 2005. URL <http://www.site.uottawa.ca/~ural/list.html>.
122. Joyner, David and William Stein. “Open Source Mathematical Software”. *Notices of the AMS*, November:1279, 2007. URL <http://www.ams.org/notices/200710/>.
123. Juillé, Hugues and Jordan B. Pollack. “A sampling-based heuristic for tree search applied to grammar induction”. *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, 776–783. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1998. ISBN 0262510987.

124. Kanazawa, Makoto. *Learnable classes of categorial grammars*. Ph.D. thesis, Stanford University, Stanford, CA, USA, 1995.
125. Kannan, Jayanthkumar, Jaeyeon Jung, Vern Paxson, and Can Emre Koksul. “Semi-automated discovery of application session structure”. *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 119–132. ACM, New York, NY, USA, 2006. ISBN 1595935614.
126. Karagiannis, Thomas, Andre Broido, Nevil Brownlee, K.C. Claffy, and Michalis Faloutsos. “Is P2P dying or just hiding?”, December 2004. URL <http://www.caida.org/publications/papers/2004/p2p-dying/>.
127. Karagiannis, Thomas, Andre Broido, Michalis Faloutsos, and Kc claffy. “Transport layer identification of P2P traffic”. *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 121–134. ACM, New York, NY, USA, 2004. ISBN 1581138210. URL <http://www.caida.org/publications/papers/2004/p2p-layerid/>.
128. Kefalas, Petros, George Eleftherakis, and Evangelos Kehris. “Communicating X-machines: a practical approach for formal and modular specification of large systems”. *Information and Software Technology*, 45(5):269–280, 2003.
129. Kent, S.. and K. Seo. “Security Architecture for the Internet Protocol”, December 2005. URL <http://tools.ietf.org/html/rfc4301>.
130. Knuth, Donald E. “backus normal form vs. Backus Naur form”. *Communications of the ACM*, 7(12):735–736, 1964. ISSN 0001-0782.
131. Kobayashi, Satoshi and Takashi Yokomori. “Learning approximately regular languages with reversible languages”. *Theoretical Computer Science*, 174(1-2):251–257, 1997. ISSN 0304-3975.
132. Koziol, Jack, David Litchfield, Dave Aitel, Chris Anley, Neel Mehta, and Riley Hassell. *The Shellcoder’s Handbook : Discovering and Exploiting Security Holes*. John Wiley & Sons, 2004. ISBN 0764544683.
133. Kreibich, Christian. “Design and Implementation of Netdude, a Framework for Packet Trace Manipulation”. *Proc. USENIX/FREENIX*. 2004.
134. Krutz, Ronald L. *Securing SCADA systems*. Wiley Pub., Indianapolis, IN, 2006. ISBN 0764597876.
135. Kunz, Thomas. *Reverse-Engineering Distributed Applications to Understand their Behaviour*. Technical Report TI-3/94, Institut für Theoretisch Informatik, Fachbereich Informatik, Technische Hochschule Darmstadt, April 1994.
136. Lai, Zhifeng, S.C. Cheung, and Yunfei Jiang. “Dynamic Model Learning Using Genetic Algorithm under Adaptive Model Checking Framework”. *qsic*, 0:410–417, 2006. ISSN 1550-6002.

137. Lang, K. J. “Random DFA’s can be Approximately Learned from Sparse Uniform Examples”. *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, 45–52. ACM, New York, N.Y., 1992.
138. Lang, Kevin J., Barak A. Pearlmutter, and Rodney A. Price. “Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm”. *ICGI ’98: Proceedings of the 4th International Colloquium on Grammatical Inference*, 1–12. Springer-Verlag, London, UK, 1998. ISBN 3540647767. The competition web site is hosted at <http://abbadingo.cs.unm.edu/>.
139. Lawrence Berkeley National Laboratory. “The Internet Traffic Archive”, 7 Nov 2000. URL <http://ita.ee.lbl.gov/>.
140. Lawrence Berkeley National Laboratory. “Bro Intrusion Detection System”, 6 December 2007. URL <http://www.bro-ids.org/>.
141. Lawrence Berkeley National Laboratory. “Bro Quick Start Guide”, 6 December 2007. URL <http://www.bro-ids.org/manuals.html>.
142. Lee, David, Dongluo Chen, Ruibing Hao, Raymond E. Miller, Jianping Wu, and Xia Yin. “Network protocol system monitoring: a formal approach with passive testing”. *IEEE/ACM Trans. Netw.*, 14(2):424–437, 2006. ISSN 1063-6692.
143. Lee, David and Krishan Sabnani. “Reverse-engineering of communication protocols”. *Network Protocols, 1993. Proceedings., 1993 International Conference on*, 208. 19–22 October 1993.
144. Lee, Edward A. and Stephen Neuendorffer. *Tutorial: Building Ptolemy II Models Graphically*. Technical Report UCB/EECS-2007-129, EECS Department, University of California, Berkeley, Oct 2007. URL <http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-129.html>.
145. Lee, Lillian. *Learning of Context-Free Languages: A Survey of the Literature*. Technical Report Harvard University Technical Report TR-12–96, 1994, Harvard University, Harvard University, Center for Research in Computing Technology, Cambridge, MA 01238, 1996.
146. Lee, Y, J.Riggle, E. Stabler T. C. Collier, and Charles Taylor. “Adaptive communication among collaborative agents: Preliminary results with symbol grounding”. *Proceedings of the Eighth International Symposium on Artificial Life and Robotics*, 149–155. 2003.
147. Leita, Corrado, Ken Mermoud, and Marc Dacier. “ScriptGen: an automated script generation tool for honeyd”. *ACSAC ’05: Proceedings of the 21st Annual Computer Security Applications Conference*, 203–214. IEEE Computer Society, Washington, DC, USA, 2005. ISBN 0769524613.

148. Levine, B. "Use of Tree Derivatives and a Sample Support Parameter for Inferring Tree Systems." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(1):25–34, 1982.
149. Lie, David, Andy Chou, Dawson Engler, and David L. Dill. "A simple method for extracting models for protocol code". *SIGARCH Comput. Archit. News*, 29(2):192–203, 2001. ISSN 0163-5964.
150. Lin, F. J., P. M. Chu, and M. T. Liu. "Protocol verification using reachability analysis: the state space explosion problem and relief strategies". *SIGCOMM Comput. Commun. Rev.*, 17(5):126–135, 1987. ISSN 0146-4833.
151. Lippmann, Richard, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. "Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation". *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, 162–182. Springer-Verlag, London, UK, 2000. ISBN 3540410856.
152. Lombardy, Sylvain, Yann Régis-Gianas, and Jacques Sakarovitch. "Introducing VAUCANSON". *Theoretical Computer Science*, 328(1-2):77–96, 2004. ISSN 0304-3975.
153. Lucas, Simon M. "Learning DFA form Noisy Samples", May 2004. URL <http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html>.
154. Ma, Justin, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. "Unexpected means of protocol inference". 313–326, 2006.
155. Mahoney, Matthew V. and Philip K. Chan. "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection", 2003.
156. Mäkinen, E. "The grammatical inference problem for the Szilard languages of linear grammars". *Inf. Process. Lett.*, 36(4):203–206, 1990. ISSN 0020-0190.
157. Margaria, Tiziana, Harald Raffelt, Bernhard Steffen, and Martin Leucker. "The LearnLib in FMICS-jETI". *ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, 340–352. IEEE Computer Society, Washington, DC, USA, 2007. ISBN 0769528953.
158. Mariani, Leonardo and Mauro Pezzé. *Inference of Component Protocols by the kBehavior Algorithm*. Technical Report LTA:2004:05, Laboratorio di Test e Analisi del Software, 2004. URL <http://www.lta.disco.unimib.it/doc/ei/PapersF.shtm>.
159. Martin, John C. *Introduction to languages and the theory of computation*. McGraw-Hill, Boston, 3rd edition, 2003. ISBN 0072322004.
160. Mathewson, Nick and Niels Provos. "libevent - an event notification library", 11 November 2007. URL <http://monkey.org/~provos/libevent/>.

161. Matz, O., A. Miller, A. Potthoff, W. Thomas, and E. Valkema. *Report on the Program AMoRE*. Technical report, 1995. URL <http://amore.sourceforge.net/>.
162. Mayo, Mike. “Learning Petri Net Models of Non-Linear Gene Interactions”. *BioSystems*, 85(1):74–82, 2005.
163. McAfee. “McAfee Virtual Criminology Report: North American Study into Organized Crime and the Internet”, July 2005. URL http://www.mcafee.com/us/research/criminology_report.
164. McCann, P.J. and S. Chandra. “PacketTypes: Abstract Specification of Network Protocol Messages”. *Proceedings of ACM SIGCOMM*, volume 30, 321–333. 2000.
165. McGregor, A., M. Hall, P. Lorier, and J. Brunskill. “Flow clustering using machine learning techniques”. C. Barakat and I. Pratt (editors), *Proc Fifth International Workshop on Passive and Active Network Measurement (PAM 2004)*, volume 3015 of *LNCS*, 205–214. Springer, Antibes Juan-les-Pins, France, 2004.
166. McHugh, John. “Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory”. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000. ISSN 1094-9224.
167. Micheel, Jörg. “NLANR PMA: Special Traces Archive”, 25 April 2005. URL <http://pma.nlanr.net/>.
168. Miclet, L. “Regular Inference with a Tail-Clustering Method”. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):737–743, 1980.
169. Miclet, Laurent and Cyrille de Gentile. *Inferring regular grammars from positive and negative samples: two optimal lattice pruning algorithms (BIG and RIG); and beam-search heuristic boosting (BRIG)*. Technical Report Publication interne n°774, Institut National de Recherche en Informatique et en Université Catholique de l’Ouest, Novembre 1993. URL <http://www.irisa.fr/centredoc/publis/PI/>.
170. Microsoft Corporation. “Remote Desktop Protocol (Windows)”, 2 July 2007.
171. Mierswa, Ingo, Michael Wurstl, Ralf Klinkenbergf, Martin Scholz, and Timm Euler. “YALE: Rapid Prototyping for Complex Data Mining Tasks”. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2006. URL <http://rapid-i.com>.
172. Mitnick, Kevin David and William L. Simon. *The art of deception: controlling the human element of security*. Indianapolis : Wiley, 2002. ISBN 0471237124.
173. Mitnick, Kevin David and William L. Simon. *The art of intrusion: the real stories behind the exploits of hackers, intruders, and deceivers*. Wiley Pub., Indianapolis, IN, 2005. ISBN 0764569597.

174. Mohri, M. and M. Nederhof. "Regular approximation of context-free grammars through transformation". 2000.
175. Molnar, Patrice Godefroid; Michael Y. Levin; David. *Automated Whitebox Fuzz Testing*. Technical Report MSR-TR-2007-58, Microsoft Research, May 2007.
176. Montoro, Massimiliano. "Cain & Abel", 26 October 2007. URL <http://www.oxid.it/cain.html>.
177. Muggleton, Stephen. *Inductive acquisition of expert knowledge*. Addison Wesley Publishing Company, 1990. ISBN 0201175614.
178. Mutz, Darren, Christopher Kruegel, William Robertson, Giovanni Vigna, and Richard A. Kemmerer. "Reverse Engineering of Network Signatures". *Proceedings of the AusCERT Asia Pacific Information Technology Security Conference*. 2005. This research was supported by the Army Research Laboratory and the Army Research Office, under agreement DAAD19-01-1-0484. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.
179. Nakamura, Katsuhiko and Masashi Matumoto. "Incremental learning of context free grammars based on bottom-up parsing and search". *Pattern Recognition*, 38:1384–1392, 2005.
180. Nederhof, M. J. "Regular approximations of CFLs: A grammatical view", 1997.
181. Nederhof, Mark-Jan. "Context-Free Parsing through Regular Approximation". Lauri Karttunen (editor), *FSMNLP'98: International Workshop on Finite State Methods in Natural Language Processing*, 13–24. Association for Computational Linguistics, Somerset, New Jersey, 1998.
182. Nederhof, Mark-Jan. "Practical experiments with regular approximation of context-free languages". *Comput. Linguist.*, 26(1):17–44, 2000. ISSN 0891-2017.
183. Niparnan, Natee. *A genetic algorithm for finite state machine inference*. Master's thesis, Chulalongkorn University, Thailand, 2002.
184. Nowak, Martin A., Natalia L. Komarova, and Partha Niyogi. "Computational and evolutionary aspects of languages". *Nature*, 417:611–617, June 2002.
185. Nucci, Antonio. "Skype Detection: Traffic Classification in the Dark", July 2006. URL <http://www.convergedigest.com/bp-c2p/bp1.asp?ID=373&ctgy>.
186. Nuzman, Carl, Iraj Saniee, Wim Sweldens, and Alan Weiss. "A compound model for TCP connection arrivals for LAN and WAN applications". *Comput. Networks*, 40(3):319–337, 2002. ISSN 1389-1286.
187. Oates, Tim, Devina Desai, and Vinay Bhat. "Learning k-Reversible Context-Free Grammars from Positive Structural Examples". *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, 459–465. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. ISBN 1558608737.

188. Oehlert, Peter. “Violating Assumptions with Fuzzing”. *IEEE Security and Privacy*, 3(2):58–62, 2005. ISSN 1540-7993.
189. Oncina, J. and P. Gracia. “A Polynomial Algorithm To Infer Regular Languages”.
190. Orebaugh, Angela D., Gilbert Ramirez, and Jay Beale. *Ethereal Network Protocol Analyzer Toolkit*. Syngress, 2006. ISBN 1597490733.
191. Ostermann, Shawn. “tcptrace”, May 2001. URL <http://jarok.cs.ohiou.edu/software/tcptrace>.
192. Pang, Ruoming, Vern Paxson, Robin Sommer, and Larry Peterson. “binpac: a yacc for writing application protocol parsers”. *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 289–300. ACM, New York, NY, USA, 2006. ISBN 1595935614. URL <http://www.icsi.berkeley.edu/pubs/networking>.
193. Parekh, Janak J., Ke Wang, and Salvatore J. Stolfo. “Privacy-preserving payload-based correlation for accurate malicious traffic detection”. *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, 99–106. ACM, New York, NY, USA, 2006. ISBN 1595935711.
194. Passerone, Roberto and James A. Rowson. “Automatic Synthesis of Interfaces between Incompatible Protocols”. 8–13. ACM, New York, NY, USA, June 1998. ISBN 0897919645.
195. Paxson, Vern. “Automated packet trace analysis of TCP implementations”. *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, 167–179. ACM, New York, NY, USA, 1997. ISBN 089791905X.
196. Peng, Wuxu and Kia Makki. “Reachability and reverse reachability analysis of CFSMs”. *Computer Communications*, 19(8):668–674, 1996.
197. Petasis, G., G. Paliouras, C. D. Spyropoulos, and C. Halatsis. “eg-GRIDS: context-free grammatical inference from positive examples using genetic search”. G. Paliouras and Y. Sakakibara (editors), *Grammatical Inference: Algorithms and Applications; 7th International Colloquium, ICGI 2004*, volume 3264 of *LNCS/LNAI*, 223–234. Springer, 2004. ISBN 978-3-540-23410-4.
198. Pezzé, Mauro and Leonardo Mariani. “kBehavior”, 2004. The source and binary files at this web site do not un-compress properly. An alternate implementation from <http://www.csc.ncsu.edu/faculty/xie/minese/mpmd/kbInference.jar> was used to evaluate the software. See <http://people.engr.ncsu.edu/txie/minese/mpmd/kbehavior.txt> for additional information.
199. Pfleeger, Charles P. and ShariLawrence Pfleeger. “Why We Won’t Review Books by Hackers”. *IEEE Security and Privacy*, 4(4):9, 2006. ISSN 1540-7993.

200. Pham, Phong H. *Traffic classification with passive measurement*. Master's thesis, University of Oslo, Department of Informatics, 2005. URL <http://research.iu.hio.no/theses/pdf/master2005>.
201. Pinker, Steen. "Formal models of language learning." *Cognition*, 7(3):217–83, 1979.
202. Pinter, Shlomit S. and Mati Golani. "Discovering workflow models from activities' lifespans". *Comput. Ind.*, 53(3):283–296, 2004. ISSN 0166-3615.
203. Pitt, L. and M. K. Warmuth. "The minimum consistent DFA problem cannot be approximated within and polynomial". *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 421–432. ACM Press, New York, NY, USA, 1989. ISBN 0897913078.
204. Pitt, Leonard and Leslie G. Valiant. "Computational limitations on learning from examples". *J. ACM*, 35(4):965–984, 1988. ISSN 0004-5411.
205. Pitt, Leonard and Manfred K. Warmuth. "Prediction-preserving reducibility". volume 41, 430–467. Academic Press, Inc., Orlando, FL, USA, 1990. ISSN 0022-0000.
206. Pla, Ferran, Antonio Molina, Emilio Sanchis, Encarna Segarra, and F. García. "Language Understanding Using Two-Level Stochastic Models with POS and Semantic Units". *TSD '01: Proceedings of the 4th International Conference on Text, Speech and Dialogue*, 403–409. Springer-Verlag, London, UK, 2001. ISBN 3540425578.
207. Pla, Ferran and Natividad Preto. "Using Grammatical Inference Methods for Automatic Part-of-Speech Tagging", 1998.
208. Postel, J. "Internet Control Message Protocol", September 1981. URL <http://tools.ietf.org/html/rfc792>.
209. Prieto, N. and E. Vidal. "Automatic learning of structural language models". *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, volume 2, 789–792. 14-17 April 1991.
210. Ptacek, Thomas, Window Snyder, Jeremy Rauch, Dave Goldsmith, and Dino Dai Zovi. "Matasano Chargin - Archive for the 'Reversing' Catagorey", 28 October 2007. URL <http://www.matasano.com/log/category/reversing>. Web Blog Matasano Chargin at <http://www.matasano.com/log/>.
211. Raffelt, Harald, Bernhard Steffen, and Therese Berg. "LearnLib: a library for automata learning and experimentation". *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, 62–71. ACM Press, New York, NY, USA, 2005. ISBN 1595931481. URL <http://jabcs.cs.uni-dortmund.de/manual/index.php/LearnLib>.
212. Reuters. "U.S. Air Force prepares to fight in cyberspace", 3 November 2006. URL <http://www.msnbc.msn.com/id/15537784/>.

213. Rey, Maxime. “Mical < Projects”, 17 Dec 2004. URL <http://www.lrde.epita.fr/cgi-bin/twiki/view/Projects/Mical>.
214. Ricca, Flippo and Mariano Ceccato. “Laboratory of Software Analysis”, 25 May 2006. URL <http://sra.itc.it/>.
215. Rieck, Konrad and Pavel Laskov. “Detecting unknown network attacks using language models”. *Proc. DIMVA*, 74–90, 2006.
216. Rieck, Konrad and Pavel Laskov. “Language mdoels for detection of unknown attacks in network traffic”. *Journal in Computer Virology*, 2(4):243–256, February 2007. URL <http://ida.first.fraunhofer.de/~rieck/>.
217. Ritter, Jordan. “ngrep - network grep”, 18 November 2006. URL <http://ngrep.sourceforge.net/>.
218. Rodger, Susan H., Jinghui Lim, and Stephen Reading. “Increasing interaction and support in the formal languages and automata theory course”. *SIGCSE Bull.*, 39(3):58–62, 2007. ISSN 0097-8418.
219. Rodgers, Susan H. “JFLAP”, 9 October 2007. URL <http://www.jflap.org/>.
220. Rosen, Eric C. “Vulnerabilities of network control protocols: an example”. *SIGCOMM Comput. Commun. Rev.*, 11(3):10–16, 1981. ISSN 0146-4833. URL <http://tools.ietf.org/html/rfc789>.
221. Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. *SIP: Session Initiation Protocol*. Technical Report RFC3261, The Internet Engineering Task Froce, Network Wroking Group, June 2002. URL <http://tools.ietf.org/html/rfc3261>.
222. Rossey, L., R. Cunningham, D. Fred, J. Rabek, R. Lippmann, J. Haines, and M. Zissman. “LARIAT: Lincoln adaptable realtime information assurance testbed”. *Aerospace Conference Proceedings, 2002. IEEE*, volume 6. 2002. ISBN 078037231X.
223. Rossi, D. and M. Mellia. “Real-Time TCP/IP Analysis with Common Hardware”. *Communications, 2006. ICC '06. IEEE International Conference on*, volume 2, 729–735. June 2006. ISBN 1424403553.
224. Rozenberg, Grzegorz and Arto Salomaa (editors). *Handbook of formal languages*. Springer, Berlin, 1st edition, 28 May 2002. ISBN 3540606491.
225. Ruiz, Enrique Vidal. “Inferencia Gramatical (IG)”, 27 July 2004. URL <http://web.iti.upv.es/~evidal/students/doct/ig/softw/>.
226. Rulot, H., N. Prieto, and E. Vidal. “Learning accurate finite-state structural models of words through the ECGI algorithm”. *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*, volume 1, 643–646. 23-26 May 1989.

227. Rulot, H. and E. Vidal. “An efficient algorithm for the inference of circuit-free automata”. 173–184, 1988.
228. Sakakibara, Yasubumi. “Efficient learning of context-free grammars from positive structural examples”. *Inf. Comput.*, 97(1):23–60, 1992. ISSN 0890-5401.
229. Saleh, Kassem and Abdulazeez Boujarwah. “Communications software reverse engineering: a semi-automatic approach”. *Information and Software Technology*, 38(6):379–390, June 1996.
230. Saleh, Kassem, R. Probert, and K. Al-Saqabi. “Recovery of CFSM-based protocol and service design from protocol execution traces”. *Information and Software Technology*, 41(11):839–852, 1999.
231. Saleh, Kassem, R. Probert, and I. Manonmani. “Recovery of communications protocol design from protocol execution traces”. *Engineering of Complex Computer Systems, 1996 Proceedings., Second IEEE International Conference on*, 265–272. October 1996.
232. Sanchis, E., F. Casacuberta, I. Galiano, and E. Segarra. “Learning structural models of subword units through grammatical inference techniques”. *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, volume 1, 189–192. 14-17 April 1991.
233. Sanders, Chris. *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems*. No Starch Press, 2007. ISBN 1593271492.
234. Schiffman, Mike. “The Libnet Packet Construction Library - The Million Packet March”, February 2007. URL <http://www.packetfactory.net/libnet/>.
235. Schimm, Guido. “Mining exact models of concurrent workflows”. *Computers in Industry*, 53(3):265–281, March 2004.
236. Schwab, Christian. “Looking behind the automation protocols”. 41. URL <http://ethernet.industrial-networking.com/articles>.
237. Sebban, Marc, Jean-Christophe Janodet, and Frédéric Tantini. “BLUE*: a Blue-Fringe Procedure for Learning DFA with Noisy Data”, 2004. URL <http://labh-curien.univ-st-etienne.fr/~janodet/>.
238. Sen, Koushik, Mahesh Viswanathan, and Gul Agha. “Learning Continuous Time Markov Chains from Sample Executions”. *QEST '04: Proceedings of the The Quantitative Evaluation of Systems, First International Conference on (QEST'04)*, 146–155. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2185-1. URL <http://www-osl.cs.uiuc.edu/~ksen/vesta/>.
239. Shakkottai, S., N. Brownlee, and K. Claffy. “A Study of Burstiness in TCP Flows”. *Passive and Active Measurement (PAM), April 2005*. 2005. URL <http://www.caida.org/publications>.

240. Shannon, Colleen, David Moore, and K. C. Claffy. "Beyond folklore: observations on fragmented traffic". *IEEE/ACM Trans. Netw.*, 10(6):709–720, 2002. ISSN 1063-6692.
241. Silva, Carlos. "Pidgin - MSNP features support in libpurple", 13 July 2007. URL <http://developer.pidgin.im/wiki/typ0>.
242. Skoudis, Edward and Tom Liston. *Counter Hack Reloaded : A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition) (Prentice Hall Series in Computer Networking and Distributed Systems)*. Prentice Hall PTR, 2005. ISBN 0131481045.
243. Socolofsky, T. and Kale C. "RFC 1180 A TCP/IP Tutorial", January 1991. URL <http://tools.ietf.org/html/rfc1180>.
244. Song, Doug. "libdnet", 19 January 2006. URL <http://libdnet.sourceforge.net/>.
245. Sonnenburg, Sören, Mikio L. Braun, Cheng Soon Ong, Samy Bengio, Leon Bottou, Geoffrey Holmes, Yann LeCun, Klaus-Robert Müller, Fernando Pereira, Carl Edward Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Pascal Vincent, Jason Weston, and Robert Williamson. "The Need for Open Source Software in Machine Learning". *J. Mach. Learn. Res.*, 8:2443–2466, 2007. ISSN 1533-7928.
246. Starkie, Brad, F. Coste, and Menno van Zaanen. "The Omphalos Context-Free Language Learning Competition", February 2004. URL <http://www.irisa.fr/Omphalos/>.
247. Steffen, Bernhard and Hardi Hungar. "Behavior-Based Model Construction". *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, 5–19. Springer-Verlag, London, UK, 2003. ISBN 3540003487.
248. Stroulia, E., M. El-Ramly, L. Kong, P. Sorenson, and B. Matichuk. "Reverse Engineering Legacy Interfaces: An Interaction-Driven Approach". *Proceedings of the Sixth Working Conference on Reverse Engineering*, 292, 1999.
249. Sutton, Michael, Adam Greene, and Pedram. Amini. *Fuzzing: brute force vulnerability discovery*. Addison-Wesley, Upper Saddle River, NJ, 2007. ISBN 0321446119.
250. Szor, Peter. *The art of computer virus research and defense*. Addison-Wesley, Upper Saddle River, NJ, 2005. ISBN 0321304543.
251. Tanase, Matthew. "IP Spoofing: An Introduction", 11 March 2003.
252. Tanenbaum, Andrew S. *Computer Networks*. Upper Saddle River, NJ : Prentice Hall PTR, 2003. ISBN 0130384887.

253. Tanenbaum, Andrew S. and Maarten Van Steen. *Distributed Systems Principles and Paradigms*. Upper Saddle River, NJ : Prentice Hall, 2007. ISBN 0132392275.
254. Tesson, Pascal and Denis Thérien. “Logic Meets Algebra: the Case of Regular Languages”. *LMCS*, 3:4, 2007.
255. Torres, Inés and Amparo Vanora. “An Efficient Representation to k -TSS Language Models”. *Computación y Sistemas*, 3(4):273–244, 2000. ISSN 1405-5546.
256. Trakhtenbrot, B.A. and Y.M. Barzdin. *Finite Automata: Behavior and Synthesis*. North-Holland, 1973.
257. Tretmans, Jan. “An overview of OSI conformance testing”, January 2001. URL <http://www.cs.aau.dk/~kgl/T0V03/>.
258. Tridgell, Andrew. “How Samba was written”, August 2003. URL http://samba.org/ftp/tridge/misc/french_cafe.txt.
259. Tridgell, Andrew. “The PFIF Agreement”, 20 December 2007. URL http://samba.org/samba/PFIF/PFIF_agreement.html.
260. Turner, Aaron. “Flowreplay Design Notes”, 2003. URL <http://tcpreplay.synfin.net/trac/wiki/flowreplayDesign>.
261. Ural, H. and H. Yenigun. *Towards Design Recovery from Observations*. Technical report, School of Information Technology and Engineering (SITE), University of Ottawa, 800 King Edward Avenue, Ottawa, Ontario, K1N 6N5, Canada, June-August 2004.
262. Valencia, Gerardo and Gabriela Coronado. “GECCO 2004”, June 2004. URL <http://isgec.org/gecco-2004/>.
263. Valiant, L. G. “A theory of the learnable”. *Communications of the ACM*, 27(11):1134–1142, 1984.
264. van de Meent, R. and A. Pras. “Assessing Unknown Network Traffic”, 19 February 2004.
265. van der Aalst, Wil and Minseok Song. “Discovering Social Networks from Event Logs”, December 2005.
266. van der Aalst, Wil, Ton Weijters, and Laura Maruster. “Workflow Mining: Discovering Process Models from Event Logs”. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004.
267. van Zijl, Lynette. “MERLin Project Page”, 2007. URL <http://www.cs.sun.ac.za/~lynette/MERLin/>.
268. Varona, A. and I. Torres. “Using smoothed K-TSS language models in continuous speech recognition”. *ICASSP '99: Proceedings of the Acoustics, Speech, and Signal Processing, 1999. on 1999 IEEE International Conference*, 729–732. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7803-5041-3.

269. Vidal, E. Rulot, J.M. H. Valiente, and G. Andreu. "Application of the error-correcting grammatical inference algorithm (ECGI) to planar shape recognition". *Grammatical Inference: Theory, Applications and Alternatives, IEE Colloquium on*. 1993.
270. Vidal, Enrique. "Grammatical Inference: An Introduction Survey". *ICGI '94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, 1–4. Springer-Verlag, London, UK, 1994. ISBN 3-540-58473-0.
271. Vidal, Enrique and David Llorens. "Using knowledge to improve N-gram language modelling through the MGCI methodology". *ICGI '96: Proceedings of the 3rd International Colloquium on Grammatical Inference*, 179–190. Springer-Verlag, London, UK, 1996. ISBN 3540617787.
272. Vorovyev, Vladimir. "trafshow", March 2006. URL <http://soft.risp.ru/trafshow/>.
273. Wilcox, Paul MBA. *Professional verification: a guide to advanced functional verification*. Kluwer Academic, Boston, 2004. ISBN 1402078757.
274. Witten, I. H. (Ian H.) and Eibe. Frank. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufman, Amsterdam, 2nd edition, 2005. ISBN 0120884070. URL <http://www.cs.waikato.ac.nz/~ml>.
275. Wojtczuk, Rafal. "Libnids", July 2007. URL <http://libnids.sourceforge.net/>.
276. Wood, Derick. "The Grail Home Page". URL <http://www.cse.ust.hk/faculty/dwood/grail>.
277. Wright, Charles V., Fabian Monrose, and Gerald M. Masson. "On Inferring Application Protocol Behaviors in Encrypted Network Traffic". *J. Mach. Learn. Res.*, 7:2745–2769, 2006. ISSN 1533-7928. URL <http://jmlr.csail.mit.edu/papers/v7/wright06a.html>.
278. Wright, Keith. "Identification of unions of languages drawn from an identifiable class". *COLT '89: Proceedings of the second annual workshop on Computational learning theory*, 328–333. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. ISBN 1558600868.
279. Wyler, Neil R., Bruce Potter, and Chris Hurley. *Aggressive Network Self-Defense*. Syngress, 2005. ISBN 1931836205.
280. Wynne, Michael W. "Speeches : Cyberspace as a Domain In which the Air Force Flies and Fights", November 2006. Remarks as delivered to the C4ISR Integration Conference, Crystal City, Va., Nov. 2, 2006.
281. Yahalom, Saar. "Tcp Session Reconstruction Tool", September 2007. URL <http://www.codeproject.com/useritems/TcpRecon.asp>.

282. Yokomori, Takashi. “Polynomial-time identification of very simple grammars from positive data”. *Theoretical Computer Science*, 298(1):179–206, 2003. ISSN 0304-3975.
283. Yokomori, Takashi and Satoshi Kobayashi. “Learning Local Languages and Their Application to DNA Sequence Analysis”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20, 1067–1079. IEEE Computer Society, Washington, DC, USA, 1998. ISSN 0162-8828.
284. Yoshinari, Tomokazu, Etsuji Tomita, and Mitsuo Wakatsuki. “Polynomial Time Identification in the Limit of Regular Languages in Some Subclass”. *IEICE technical report. Theoretical foundations of Computing*, 101(184):49–56, 2001. ISSN 09135685. URL <http://ci.nii.ac.jp/naid/110003191915/en/>.
285. Yu, Sheng. “Grail+ A symbolic computation environment for finite-state machines, regular expressions, and finite languages.”, 2002. URL <http://www.csd.uwo.ca/Research/grail/>.
286. Zissman, Marc. “MIT Lincoln Laboratory - DARPA Intrusion Detection Evaluation”, 2001. URL <http://www.ll.mit.edu/IST/ideval>. Dr. Marc Zissman Lincoln Laboratory Massachusetts Institute of Technology 244 Wood Street Lexington, MA 02420-9108 Phone (781) 981-5500 intrusion@sst.ll.mit.edu.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 27-03-2008		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) May 2006 — Mar 2008		
4. TITLE AND SUBTITLE Dynamic Protocol Reverse Engineering A Grammatical Inference Approach				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) DeYoung, Mark E., Capt, USAF				5d. PROJECT NUMBER JON#08-159		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management AFIT/EN 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/08-06		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Mr. Robert J. Kaufman, III AFIOC/IO 102 Hall Blvd, Suite 345 Lackland AFB, TX 78236 Comm (210) 977-5377 robert.kaufman@lackland.af.mil				10. SPONSOR/MONITOR'S ACRONYM(S)		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT Round trip engineering of software from source code and reverse engineering of software from binary files have both been extensively studied and the state-of-practice have documented tools and techniques. Forward engineering of protocols has also been extensively studied and there are firmly established techniques for generating correct protocols. While observation of protocol behavior for performance testing has been studied and techniques established, reverse engineering of protocol control flow from observations of protocol behavior has not received the same level of attention. State-of-practice in reverse engineering the control flow of computer network protocols is comprised of mostly ad hoc approaches. We examine state-of-practice tools and techniques used in three open source projects: Pidgin, Samba, and rdesktop. We examine techniques proposed by computational learning researchers for grammatical inference. We propose to extend the state-of-art by inferring protocol control flow using grammatical inference inspired techniques to reverse engineer automata representations from captured data flows. We present evidence that grammatical inference is applicable to the problem domain under consideration.						
15. SUBJECT TERMS Grammatical Inference, Protocol Reverse Engineering						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert F. Mills, PhD, AFIT/ENG	
U	U	U	UU	167	19b. TELEPHONE NUMBER (include area code) (937) 785-3636, ext 4527 robert.mills@afit.edu	